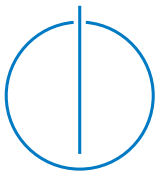
FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Development of a benchmark suite for Uncertainty Quantification of laminar flow

Christian Ziegner



FAKULTÄT FÜR INFORMATIK

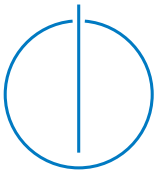
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Development of a benchmark suite for Uncertainty Quantification of laminar flow

Entwicklung einer Benchmark-Suite für Uncertainty Quantification laminarer Strömungen

Author:	Christian Ziegner
Supervisor:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	Dr. Tobias Neckel
Submission Date:	October 15, 2014



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, October 15, 2014 Christian Ziegner

Acknowledgments

At first, I want to thank my supervisor Prof. Dr. Hans-Joachim Bungartz for giving me the opportunity to explore this highly interesting topic.

Further, I would like to thank my advisor Dr. Tobias Neckel for suggesting this topic to me and for always helping me out with great guidance and kind words, although being constantly very busy.

I want to thank Christoph Kowitz for providing me with all the necessary support and information about the CFD lab code and other programming issues.

Additionally, I want to thank my friends, especially Nils, for constantly pushing me forward and helping me revise my first draft in the end.

Last, but surely not least, I would like to thank my girlfriend, Rebecca, for always cheering me up and bearing my whining when things did not go so well. You are the best!

Abstract

Uncertainty Quantification is a special approach to analyzing complex systems or algorithms. It takes input errors into account, in order to gain knowledge about the outcome of a system. In the last few years, the consciousness about the potential and importance of uncertainty quantification methods has rapidly increased, but, as everything else, it needs time to develop. This thesis aims to support and even improve the development of uncertainty quantification methods by establishing a tool chain consisting of DAKOTA, a state of the art analyzing and optimization framework, and a complex simulation code from the area of computational fluid dynamics. It discusses the coupling of the two programs in detail and performs several analyses on the simulation code, using a Monte Carlo sampling and a stochastic collocation method. In the end, this work provides a mean for analyzing newly developed uncertainty quantification methods by generating a composition of benchmark data gained from the uncertainty quantification studies on the simulation code. Besides the data that can be used for comparison, it supplies developers with the corresponding test framework as well. With a simple interface, based on reading and writing files for data exchange, users can integrate their own uncertainty quantification methods without great effort.

Contents

Acknowledgments **iii**

Abstract **v**

1 Introduction **1**
1.1 Intention and Content of this Thesis 1
1.2 Thesis Structure 2

2 Computational Fluid Dynamics **3**
2.1 Introduction 3
2.2 Mathematical Description of incompressible Flow 4
2.2.1 The Navier-Stokes-Equations 5
2.2.2 Boundary Conditions 6
2.3 CFD Lab Code 7
2.3.1 Discretization of the Navier-Stokes-Equations 7
2.3.2 The Algorithm 8
2.3.3 Description of the Code Interface 10

3 Uncertainty Quantification **13**
3.1 Introduction 13
3.2 Probabilistic Framework 13
3.3 Methods 14
3.3.1 Monte Carlo Sampling 14
3.3.2 Stochastic Collocation 14
3.4 DAKOTA 16
3.4.1 Installation 17
3.4.2 Interaction with the Framework 19

4 Benchmark-Suite **25**
4.1 Benchmark Scenarios 25
4.1.1 Backward Facing Step 25
4.1.2 Channel with Circular Obstacle 26
4.2 Preparation of the CFD Lab Code 28
4.2.1 Structure of the Output 28
4.2.2 Channel with Circular Obstacle 30
4.2.3 Reattachment Length 31

4.2.4 Drag and Lift Coefficients . . . 31
4.2.5 Termination Condition . . . 32
4.3 Coupling of the CFD Lab Code with DAKOTA . . . 32
4.3.1 Wrapper Scripts . . . 33
4.3.2 DAKOTA Input Files . . . 37
4.3.3 Running the UQ Experiments . . . 41

5 Numerical Results **43**
5.1 Backward Facing Step . . . 43
5.1.1 Deterministic Results . . . 43
5.1.2 UQ Results . . . 44
5.2 Channel with Circular Obstacle . . . 45
5.2.1 Deterministic Results . . . 46
5.2.2 UQ Results . . . 47

6 Conclusion and Outlook **51**

List of Figures **53**

List of Tables **55**

Bibliography **57**

1 Introduction

In a wide range of scientific and engineering applications, such as structural mechanics, heat transfer and fluid dynamics, computational simulation systems are often used as prototypes to understand the underlying complex physics. So far, in numerical simulations, most effort has been devoted to the development of highly accurate algorithms where numerical errors are small and under control. However, due to incomplete knowledge of the underlying physics and inevitable measurement errors, uncertainty is ubiquitous. Consequently, it is not possible to construct real life experiments of large-scale complex problems that are completely free of errors. Sources of errors for such complex problems are, for example, initial and boundary uncertainty, measurement inaccuracies, interpolation errors and geometrical deviation. It is virtually impossible to determine parameter values of real life phenomena as precisely as it is assumed in most numerical simulations. Since even small input errors can result in non-negligible effects on the output (see [XK04]), fluctuations in parameters of the physical system have to be taken into account, in order to generate a realistic computer simulation.

One approach to solving this problem is the concept of uncertainty quantification (UQ), of which theoretical and practical importance has increased rapidly in the last few years. The primary goal of UQ is to quantify the impact that errors in input parameters have on the output. It tries to determine the likelihood of certain outcomes if several aspects of the system are not exactly known. Furthermore, it can help to understand which uncertain inputs contribute most to the variance of the output. The great benefit of uncertainty quantification is that outcomes of real life problems can be predicted more accurately with the help of numerical simulations.

1.1 Intention and Content of this Thesis

Approaches relating to the UQ topic are still developing at a very fast pace. So far, a large variety of uncertainty quantification methods are available. However, one can barely find reliable research data to compare newly developed techniques with, at the present. Therefore, it is reasonable to pick up existing methods and evaluate them in conjunction with realistic problems.

In this thesis, we will evolve a composition of benchmarks by coupling a professionally developed UQ framework with different scenarios from the area of fluid dynamics. We will further provide a ready to use tool chain with this thesis, of which developers can replace the UQ component by their own method and instantly test and compare it. Evaluating flow scenarios address a realistic process, since those problems are nonlinear

and, due to their complexity, rife with sources of errors. Additionally, most fluid algorithms are computationally intensive, so that it is unlikely that the utilized UQ methods are incorrectly assessed as efficient and highly accurate. As the UQ component of our benchmark suite we use DAKOTA[1], a state of the art analyzing framework developed by professionals at the United States Department of Energy.

1.2 Thesis Structure

The paper is structured as follows: We will start off with a general overview on the field of computational fluid dynamics (CFD), as we are using a CFD simulation code as the numerical solver in our tool chain. Additionally, we will go into the mathematical background of the simulation code and give some basic information about it. In the third chapter, a further description of the UQ topic and the two utilized methods will be given, followed by a characterization of the DAKOTA framework and its possibilities. Afterwards, we will shortly explain the analyzed flow scenarios and state their configuration. Since we had to modify the simulation code to achieve the desired behavior, we will also list our additional implementations. In the end of this chapter, we will describe how we integrated the simulation code into the UQ framework. The fifth chapter will state the results of the different UQ studies and shortly evaluate it. These results should represent reference or benchmark data that can be used for comparing the outcome of newly developed UQ methods and, thereby, estimating their quality. In order to validate the functionality of our algorithms, we will first compare deterministic results to those of referenced literature. In this comparison, small deflections are noticeable, which are of no importance here, as we do not want to provide data for physical evaluations but rather a proof of concept. To round things off, we will come to a conclusion in the last chapter and give an outlook on further possibilities that pick up where this bachelor's thesis coverage ends.

[1] Project page: `http://dakota.sandia.gov/`

2 Computational Fluid Dynamics

In the following chapter, we will describe the general physical background of computational fluid dynamics (CFD). Afterwards, we will introduce the simulation code that is used as the numerical solver in the simulation runs. There will be an illustration of the mathematical description of flows and how its discretization is implemented in the simulation code. Furthermore, we will shortly describe the basic algorithm of the solver. Lastly, there will be further information about the possibilities of the code, as well as explanations on how to communicate with it.

2.1 Introduction

Let us start off with a short introduction to fluid dynamics. Since CFD is not the key part of this thesis, there will only be given some general information about fluids and their approaches here. If you want to go into more detail, we recommend to have a close look at [GDN98] or [KCD11].

There exist two entirely different approaches for the description of fluid motion. These are the Eulerian approach on the one hand and the Lagrangian approach on the other. According to Euler, you examine properties of interest, such as velocity and pressure, at certain fixed locations in the field of the flow. You can imagine this as a window revealing a view on all of the characteristics of the fluid at that certain area. As opposed to the Eulerian approach, the Lagrangian examines this data by focusing on each individual particle that moves inside the flow field. You can say that you follow a particle on its way through the flow field and detect potential changes of its properties. The choice of which approach to use highly depends on the simulation scenario. For example, if you want to examine the properties of an airplane wing in a wind channel, you are not interested in the properties of particles before and after "interaction" with the wing. You rather want to observe the activity close by the wing itself. Therefore, the Eulerian approach would be preferred in this case.

We will now address the characteristics of fluids in more detail. Generally, a fluid is defined either as a liquid or a gas. The most important characteristic of a fluid is called its viscosity, which can be described as the resistance of a fluid to a change in shape, either caused by shear stress or extensional stress. On the particle level you could say that viscosity is the resistance to movement of neighboring particles relative to one another. It can be seen as an internal friction between the molecules of a fluid, which leads to the fact, that the development of velocity can differ within a fluid. The following examples can be given for fluids of low, medium and high viscosity:

- low viscosity: helium, gases in general
- medium viscosity: water, milk, alcohol
- high viscosity: honey, syrup, glue, toothpaste

Another important property is the compressibility of fluids. This leads to a distinction between two types: Fluids of which density can change due to external pressure are called compressible fluids. If this property is not existent, the fluid is called incompressible. Every gas is highly compressible, liquids are not. Most calculations done in fluid dynamics assume that the fluid is incompressible, which is acceptable for most fluids (except gases), as their compressibility is very low and can be ignored.

For the description of a fluid's flow as a whole, a dimensionless quantity exists, called the Reynolds number (Re). The Re is one of the most important quantities in the field of fluid dynamics. It was established by Osborne Reynolds who defined it as follows:

$$Re = \frac{\rho v L}{\mu} \ , \tag{2.1}$$

where ρ stands for the density of the corresponding fluid, v is the characteristic velocity, e.g. the inflow velocity in a flow channel scenario, L is the characteristic length, e.g. the length of an obstacle inside of a flow channel, and μ is the dynamic or absolute viscosity of the corresponding fluid.

The Reynolds number expresses the ratio of inertial forces to viscous forces and can be used to determine characteristics of the flow behavior. If the Reynolds number of a flow is small (Re « 2000), it is considered as laminar flow. In laminar flow the fluid travels in regular layers, e.g. parallel to the walls of a channel, that are not very likely to mix. Usual laminar flow is either slow or consists of a highly viscous fluid. The opposite of laminar flow is turbulent flow, which is characterized by a high Reynolds number (RE » 4000). In turbulent flow the fluid takes twirling paths and the occurrence of mixing is very likely. Most turbulent flow is fast and features low viscous fluids. Figure 2.1 illustrates the difference of these two types of flow.

In this paper we only consider the Eulerian approach for describing fluid-motion of incompressible, laminar flow in two-dimensional scenarios.

2.2 Mathematical Description of incompressible Flow

In the following, we will introduce the mathematical description of incompressible, laminar flow. We will first state the important quantities for the flow of a fluid in the domain $\Omega \subset \mathcal{R}^d$, with $d \in \{2, 3\}$, in the time interval $[t_0; t_{end}] \subset \mathcal{R}_0^+$, here:

- $\vec{u} \in \mathbb{R}^d$: velocity vector
- p: pressure

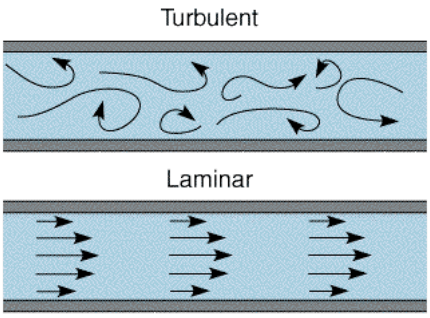


Figure 2.1: Difference between laminar and turbulent flow. (Source: `http://blog.nialbarker.com/252/slow_is_faster`)

- ρ: density

Since we solely consider incompressible flow, of which density does not change over time, we have $\rho(\vec{x}, t) = \rho_\infty = const.$, with $\vec{x} \in \Omega$.

2.2.1 The Navier-Stokes-Equations

The Navier-Stokes-Equations are partial differential equations that describe incompressible flow in a mathematical way. They consist of the momentum equation

$$\frac{\partial}{\partial t}\vec{u} + (\vec{u} \cdot \nabla)\vec{u} + \frac{1}{\rho_\infty}\nabla p = \frac{\mu}{\rho_\infty}\triangle\vec{u} + \vec{g} \ , \tag{2.2}$$

where μ is the dynamic viscosity and g stands for external forces, like gravity, and the continuity equation

$$\nabla \cdot \vec{u} = 0 \ . \tag{2.3}$$

For a description of the derivation of these equations, we recommend reading [GDN98] or [Nec09].

In order to carry over findings from numerical simulations directly to real world phenomena, the dimensionless form of (2.2) and (2.3) has to be used. To do so, one needs to transfer their quantities into dimensionless quantities:

$$\text{dimensionless quantity} = \frac{\text{dimensional quantity}}{\text{reference quantity with the same physical unit}}$$

The dimensionless variables for the Navier-Stokes-Equations are:

$$\begin{aligned} x^* &:= \frac{\vec{x}}{L_\infty} \\ \vec{u}^* &:= \frac{\vec{u}}{\vec{u}_\infty} \\ p^* &:= \frac{p - p_\infty}{\rho_\infty u_\infty^2} \\ t^* &:= \frac{t}{T_\infty} = t\frac{u_\infty}{L_\infty} \end{aligned}$$

$L_\infty, u_\infty, p_\infty$ and T_∞ are reference quantities, like obstacle length, inflow velocity, etc. By additionally introducing the dimensionless external forces $\vec{g}^* := \frac{L}{u_\infty^2}\vec{g}$ and with definition (2.1), we obtain

$$\frac{\partial}{\partial t^*}\vec{u}^* + (\vec{u}^* \cdot \nabla^*)\vec{u}^* + \nabla^* p^* = \frac{1}{Re}\triangle^*\vec{u}^* + \vec{g}^* \tag{2.4}$$

and

$$\nabla^* \cdot u^* = 0 \tag{2.5}$$

as the dimensionless form of the Navier-Stokes-Equations (2.2) and (2.3). Here ∇^* and $\triangle^*$ are the same operators as those in the dimensional equations, but referring to $\vec{x}^*$ rather than $\vec{x}$.

For a more detailed description, see [GDN98] or [Nec09].

2.2.2 Boundary Conditions

In addition to the equations declared above, several rules for treating points along the fixed boundary $\Gamma := \partial\Omega$ exist:

1. **No-slip condition:** Both velocity components are zero at a solid boundary.

2. **Free-slip condition:** At a solid boundary, the velocity normal to it is zero. However, there are no frictional losses at the boundary.

3. **Inflow condition:** Both velocity components are given.

4. **Outflow condition:** Normal gradients for both velocity components are zero.

5. **Periodic boundary condition:** For periodic problems, the velocity and pressure must coincide at the left and right boundaries of the periods.

2.3 CFD Lab Code

With the mathematical background of the description of a fluid-flow, we can move forward to the introduction of our simulation code. The code that we use as the deterministic solver in our tool chain is a CFD implementation that was developed at the Chair of Scientific Computing of TUM's informatics department. The implementation follows the instructions given in [GDN98] and provides numerical simulations for several flow scenarios. In the following section we will briefly sketch the discretization of the Navier-Stokes-Equations (2.4) and (2.5) as it is implemented in the CFD lab code. Note that we only consider the discretization for the two-dimensional case. Further, we will describe the basic algorithm that is followed by the code. After the explanation of the underlying implementation we will outline how the interaction between user and simulation code works in general.

2.3.1 Discretization of the Navier-Stokes-Equations

The original Navier-Stokes-Equations (2.4) and (2.5) are continuous in time and space. In order to compute numerical solutions for these nonlinear partial differential equations, discretization of the time and space derivatives is necessary. In our CFD lab code, the spatial discretization is separated from and done before the time discretization.

Discretization of the spatial derivatives For discretizing the spatial derivatives, the field of the flow is represented as a staggered regular grid consisting of equidistant cells. The cell width and height is denoted as δx and δy. In a staggered grid, the different unknown variables are not located at the same grid points. In the grid that we use, the pressure p is stored at the center of the cells, whereas the horizontal velocity u is located in the midpoint of the right edge and the vertical velocity v is located in the midpoint of the top edge of each cell. You can imagine three different grids, one for each of the unknown variables, that are shifted by half a cell to the right and half a cell to the top. The cell (i, j) occupies the region $[(i-1)\delta x, i\delta x] \times [(j-1)\delta y, j\delta y]$. Hence, the pressure $p_{i,j}$ is located at $((i-0.5)\delta x, (j-0.5)\delta y)$, the horizontal velocity $u_{i,j}$ at $(i\delta x, (j-0.5)\delta y)$ and the vertical velocity $v_{i,j}$ at $((i-0.5)\delta x, j\delta y)$. Figure 2.2 illustrates the alignment of the different variables on the grid. Unfortunately, there comes a problem along with this kind of representation, because there are no vertical velocities existent at the vertical edges. Same applies to the horizontal velocities at the horizontal edges. However, these edge values are necessary to satisfy the boundary conditions. Therefore, an extra layer of so called "ghost cells" is added around the flow domain (see Figure 2.3). With the help of these "ghost cells", values for the velocities at the edges can be calculated by interpolation.

In order to discretize the spatial derivatives, the Finite Difference Method (FDM) is used, in which the derivatives are approximated by difference quotients:

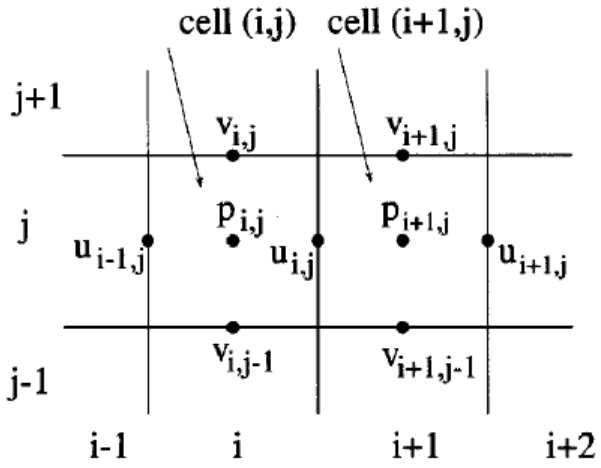


Figure 2.2: Alignment of the variables on the staggered grid. (Source: [GDN98])

$$\left[\frac{\partial u}{\partial x}\right]_{i,j} = \frac{u_{i,j} - u_{i-1,j}}{\delta x} \quad , \left[\frac{\partial v}{\partial y}\right]_{i,j} = \frac{v_{i,j} - v_{i-1,j}}{\delta y} \quad . \tag{2.6}$$

Discretization of the time derivatives For the discretization of the time derivatives, the time interval $[0, t_{\text{end}}]$ is divided into $m-1$ equal subintervals, which means that values of the variables u, v and p are computed only at $n\delta t$, with $n = 0, ..., m$. For discretizing the time derivatives at time t_{n+1}, the code makes use of the implicit Euler method:

$$\left[\frac{\partial u}{\partial t}\right]^{(n+1)} := \frac{u^{(n+1)} - u^{(n)}}{\delta t} \quad , \qquad \left[\frac{\partial v}{\partial t}\right]^{(n+1)} := \frac{v^{(n+1)} - v^{(n)}}{\delta t} \quad . \tag{2.7}$$

2.3.2 The Algorithm

In order to present the general time loop of the algorithm, we begin by applying the time discretization mentioned above in the momentum equation (2.4):

$$\begin{aligned} u^{(n+1)} &= u^{(n)} + \delta t \left[\frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \frac{\partial (u^2)}{\partial x} - \frac{\partial (uv)}{\partial y} + g_x - \frac{\partial p}{\partial x}\right] \quad , \\ v^{(n+1)} &= v^{(n)} + \delta t \left[\frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) - \frac{\partial (uv)}{\partial x} - \frac{\partial (v^2)}{\partial y} + g_y - \frac{\partial p}{\partial y}\right] \quad . \end{aligned} \tag{2.8}$$

By assigning all variables, except the pressure, to F and G:

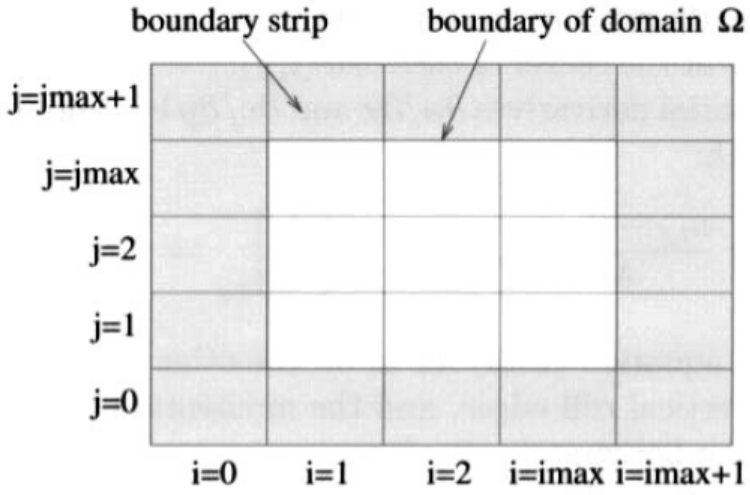


Figure 2.3: Domain with a boundary strip of "ghost cells". (Source: [GDN98])

$$
\begin{aligned}
F &= u^{(n)} + \delta t \left[\frac{1}{Re} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \frac{\partial (u^2)}{\partial x} - \frac{\partial (uv)}{\partial y} + g_x \right] \ , \\
G &= v^{(n)} + \delta t \left[\frac{1}{Re} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \frac{\partial (uv)}{\partial x} - \frac{\partial (v^2)}{\partial y} + g_y \right] \ ,
\end{aligned}
\tag{2.9}
$$

we separate the pressure derivatives and get the "semi time discrete" momentum equations in both dimensions of the form

$$
\begin{aligned}
u^{(n+1)} &= F - \delta t \frac{\partial p}{\partial x} \ , \\
v^{(n+1)} &= G - \delta t \frac{\partial p}{\partial y} \ .
\end{aligned}
\tag{2.10}
$$

To obtaining a fully time discrete version, the terms on the right hand side of (2.10) must also be assigned to a distinct time level: we associate F and G with time level n, i.e. all velocities will be evaluated at t_n, while the derivatives of the pressure p belongs to time level $(n+1)$. This gives us the fully time discrete momentum equations

$$
\begin{aligned}
u^{(n+1)} &= F^{(n)} - \delta t \frac{\partial p^{(n+1)}}{\partial x} \ , \\
v^{(n+1)} &= G^{(n)} - \delta t \frac{\partial p^{(n+1)}}{\partial y} \ .
\end{aligned}
\tag{2.11}
$$

This form of discretization can be considered as explicit in velocities and implicit in pressure, which means that, in every time step, the velocity field can only be computed if the pressure of the same time step is already known before.

The computation of the pressure should be realized with help of the continuity equation (2.5). After inserting the equations of the velocities (2.11) into the continuity equation (2.5), we convert the resulting equation and obtain a Pressure Poisson Equation (PPE) for $p^{(n+1)}$:

$$\frac{\partial^2 p^{(n+1)}}{\partial x^2} + \frac{\partial^2 p^{(n+1)}}{\partial y^2} = \frac{1}{\delta t}\left(\frac{\partial F^{(n)}}{\partial x} + \frac{\partial G^{(n)}}{\partial y}\right) \quad . \tag{2.12}$$

Therefore, time step $n+1$ consists of the following three steps:

1. Calculate $F^{(n)}$ and $G^{(n)}$ according to (2.9) from the velocities $u^{(n)}$ and $v^{(n)}$.
2. Solve the PPE (2.12) to get the pressure $p^{(n+1)}$.
3. Calculate the new velocities $u^{(n+1)}$ and $v^{(n+1)}$ using (2.8) with the pressure $p^{(n+1)}$ from step 2.

The above time loop starts at $t = 0$ and is repeated until $t = t_{\text{end}}$. For solving the equations of the time loop, the CFD lab code makes use of the PETSc libraries (see [Bal+14b], [Bal+14a] and [Bal+97]).

The procedure for obtaining the PPE is known as the Chorin projection method (see [GDN98]).

2.3.3 Description of the Code Interface

Apart from the mathematical background about flow description and its implementation in our simulation code, there are a few more important considerations concerning the handling of the code, that will be explained in the following section.

For running the simulation code, you need to provide a valid configuration file with all necessary parameter specifications, e.g. geometry variables and Reynolds number. In Figure 2.4, a sample configuration file can be examined. With a feasible configuration file called "config.xml", the code can be executed with the command

```
./ns config.xml
```

within the directory of the simulation code. The file `ns` is the executable program of the simulation. During the simulation, the code produces several .vtk files. You can visualize the simulation by opening these files with a visualization tool. Figure 2.5 illustrates an excerpt of a simulation run visualized by Paraview[2].

The original version of the code offers a variety of features and different flow scenarios like driven cavity, backward facing step, etc. In Section 4.1 we will further describe the two scenarios that we used in our analysis. Unfortunately, some of the key functions that we need for our benchmark suite were missing in the raw code. In Section 4.2 we will examine the most important modifications.

[2] `http://www.paraview.org/`

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <flow Re="100" />
    <simulation finalTime="1000" epsilon="0.000001">
        <type>dns</type>
        <scenario>channel</scenario>
    </simulation>
    <backwardFacingStep xRatio="0.12" yRatio="0.5" />
    <timestep dt="0.5" tau="0.5" />
    <solver gamma="0.5" />
    <geometry
      dim="2"
      dx="0" dy="0" dz="0"
      sizeX="200" sizeY="20" sizeZ="0"
      lengthX="100" lengthY="10" lengthZ="1" >
      <mesh>uniform</mesh>
        <!--<mesh>stretched</mesh>-->
    </geometry>
    <environment gx="0" gy="0" gz="0" />
    <walls>
        <left>
            <vector x="1.0" y="0" z="0" />
        </left>
        <right>
            <vector x="0" y="0" z="0" />
        </right>
        <top>
            <vector x="0" y="0" z="0" />
        </top>
        <bottom>
            <vector x="0" y="0" z="0" />
        </bottom>
        <front>
            <vector x="0" y="0" z="0" />
        </front>
        <back>
            <vector x="0" y="0" z="0" />
        </back>
    </walls>
    <vtk interval="0.1">channel_result</vtk>
    <stdOut interval="0.01" />
    <parallel numProcessorsX="1" numProcessorsY="1" numProcessorsZ="1" />
</configuration>
```

Figure 2.4: Sample configuration file for a two-dimensional backward facing step scenario (scenario = channel) with $Re = 100$. The length (lengthX) is 100 and divided into 200 cells (sizeX). The height (lengthY) is 10 and divided into 20 cells (sizeY). The inflow velocity vector is specified as 1.0 in x direction (x component of the vector for the left wall). The size of the backward facing step is defined as 12% of the channel length in x direction and 50% of the channel height in y direction.

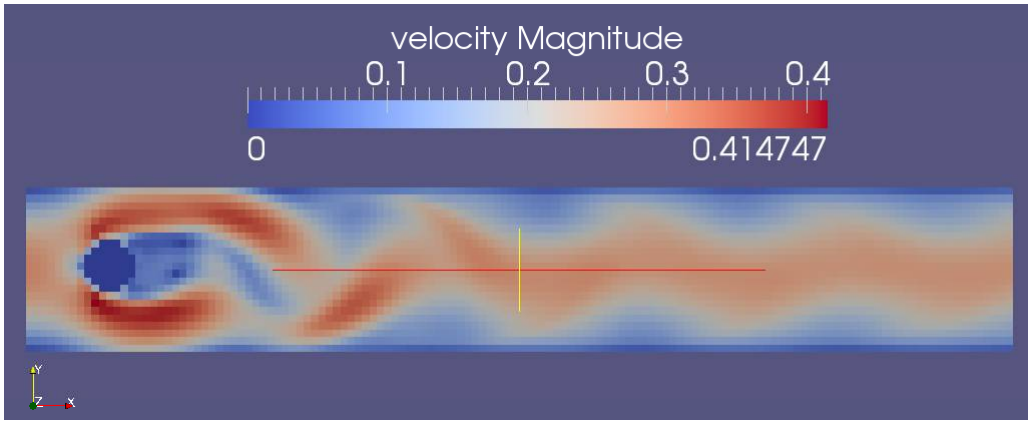


Figure 2.5: Sample Paraview visualization of a flow scenario with a circular obstacle. Each cell of the flow field is colored according to its velocity magnitude.

3 Uncertainty Quantification

In this chapter we will briefly explain the idea of uncertainty quantification and shortly describe the functionality of two commonly used UQ methods, Monte Carlo sampling (MCS) and stochastic collocation (SC). Afterwards, we will present the framework DAKOTA which offers, among many other analyzing methods, several UQ methods. This framework was used for the UQ part of our benchmark suite computations. We will first describe some overall features of DAKOTA and then offer instructions on how to install the framework. At the end of this chapter, the interface of DAKOTA will be described in detail.

3.1 Introduction

The general idea of uncertainty quantification is to estimate the effect of input inaccuracies on results of real life experiments. The importance of understanding such uncertainties has been realized by many researchers for a long time. Up to the present, a lot of methods have been developed. The most dominant approach is to treat uncertainties as random variables or random processes and regard the original deterministic systems as stochastic systems. As mentioned before, the most common methods are Monte Carlo and stochastic collocation which we will describe later in this chapter. More information about UQ can be found in [Smi13], [Xiu10] and [Xiu09].

3.2 Probabilistic Framework

In this section we will establish the underlying probabilistic framework that will be used in the descriptions of the UQ methods. Note that we are only focusing on continuous random variables.

Our probability space is defined as $(\Omega, \mathcal{A}, P)$, where Ω denotes the event space, $\mathcal{A}$ the σ-algebra on the set Ω and P the probability measure on $\mathcal{A}$. We are using the random variable $X : \Omega \to \mathbb{R}$ as a representation of a random input parameter. X is a function that assigns a probability to every event $\omega \in \Omega$ that may occur. For N random input parameters and the corresponding N random variables, the function definition extends to $Y = [X_1, ..., X_N] : \Omega^N \to \mathbb{R}^N$.

For evaluation we have to solve $Z = u(X_1, ..., X_N) = u(Y)$, where Z is another random variable, that represents simulation output that we are interested in, and $u : \mathbb{R}^N \to \mathbb{R}$ is the solution of the partial differential equation (PDE) that is solved numerically. After solving the PDE, we are interested in several statistical quantities, such as mean and

variance, that can be used as tools to predict characteristics of the outcome. Their computation will be mentioned in the method descriptions below.

Since errors that may occur for physical parameters in real life can, in general, be represented by a normal distribution very accurately, we will consider the random input variables X_i to be normally distributed in the following, i.e. $X_i \sim \mathcal{N}(\mu_i, \sigma_i)$, where μ_i is the mean $\mathbb{E}[X_i]$ and σ_i the standard deviation σ_{X_i} of the random variable X_i.

3.3 Methods

In this section we will describe two of the existing UQ methods. For descriptions of other methods, see [Smi13], [Xiu10] or [Xiu09].

3.3.1 Monte Carlo Sampling

Monte Carlo sampling is a simple and often used UQ method. It represents a highly effective sampling approach to uncertain quantification. In MCS, Q multiple independent realizations Z_i, with $i = 1, ..., Q$, of the simulation are generated. The random input Y_i, with $i = 1, ..., Q$, is based on the corresponding probability distribution. Due to the fact that for each realization the data is fixed, the problem becomes deterministic. By running the solver multiple times we gain a collection of Q different output values. After all of the required samplings have been acquired, statistical information, such as mean

$$\mathbb{E}[Z] \approx \frac{1}{Q} \sum_{i=1}^{Q} u(Y_i) = \frac{1}{Q} \sum_{i=1}^{Q} Z_i \tag{3.1}$$

and variance

$$Var(Z) \approx \frac{1}{Q-1} \sum_{i=1}^{Q} (Z_i - \mathbb{E}[Z])^2 \ , \tag{3.2}$$

can be extracted from the ensemble of solutions. Despite the straightforward implementation and appliance of this method, it is not recommended to use it for computationally intensive systems, because it typically requires a large number of executions for the statistical quantities to converge. For example, the mean value converges with $\mathcal{O}(\frac{1}{\sqrt{Q}})$. In order to accelerate the convergence of UQ sampling methods, many modifications have been developed, e.g. Latin Hypercube Sampling and Quasi-Monte Carlo methods. However, we will not take them into account in this paper.

3.3.2 Stochastic Collocation

Stochastic collocation methods are more complex and less computationally intensive methods for evaluating the problem described above. In general, collocation methods

are defined as algorithms that solve the PDE at discrete points of the corresponding input space. Additionally, the term stochastic collocation in UQ focuses on techniques that make use of polynomial approximation theory to strategically locate the collocation points, also called "nodes", in order to gain accurate results with little computational effort. Two of the major approaches of high-order stochastic collocation methods are the Lagrange interpolation and the pseudo-spectral generalized polynomial chaos (gPC). In the following we will shortly explain the functionality of the pseudo-spectral approach.

Basically, we want to obtain an approximate P-th order gPC projection for the random input vector Y

$$w_P(t,x;Y) \approx \sum_{m=1}^{M} \hat{w}_m(t,x)\phi_m(Y) \ \text{, with } \ M = \binom{N+P}{N} \ , \tag{3.3}$$

where N is the number of the independent random input variables and $\{\phi_m(Y)\}$ denotes a set of orthogonal polynomials with polynomial order of at most P. The polynomials are determined by the probability distribution of the random input variables in Y (see [Xiu09]). We assume that $\mathbb{I} \in \mathbb{R}^N$ is the input space for Y. Then the gPC expansion coefficients are determined as

$$\hat{w}_m(t,x) = \int_{\mathbb{I}} u(Y)\phi_m(Y)\,\mathrm{d}Y \ \text{, with } \ m = 1,...,M \ , \tag{3.4}$$

which can be approximated with the previously determined set of Q quadrature nodes and according quadrature weights $\{Y_j, \alpha_j\}_{j=1}^{Q}$ by

$$\hat{w}_m(t,x) \approx \sum_{j=1}^{Q} u(t,x;Y_j)\phi_m(Y_j)\alpha_j \ \text{, with } \ m = 1,...,M \ . \tag{3.5}$$

The term $u(t,x;Y_j)$ in this equation stands for the PDE with fixed parameter Y_j that can be solved by e.g. a deterministic solver.

With the formulations specified above, we can now state the general algorithm for a stochastic collocation method using the pseudo-spectral gPC approach:

1. Generate a nodal set $\{Y_j, \alpha_j\}_{j=1}^{Q}$
2. Solve $u(t,x;Y_j)$ in (3.5) for each $j = 1,...,Q$
3. Evaluate the approximate gPC expansion coefficients (3.4)
4. Construct the P-th order gPC approximation (3.3)

The most important statistical quantities, mean and variance, can be calculated with

$$\mathbb{E}[u] \approx \hat{w}_1 \tag{3.6}$$

and

$$Var(u) \approx \sum_{m=2}^{M} \hat{w}_m^2 \ , \tag{3.7}$$

as described in [Xiu09].

As for all uncertainty quantification methods, the selection of the quadrature nodes is the most important ingredient in this process, since it directly influences the number of executions of the deterministic solver. It is straightforward in a one-dimensional space, but can be difficult for more dimensions. Therefore, attention has to be payed to the choice of the appropriate quadrature rule, e.g. Gaussian quadrature, sparse grid, etc., and its quadrature order.

3.4 DAKOTA

This chapter is about the framework DAKOTA that we are using for the UQ part of our benchmark suite. The acronym DAKOTA stands for Design Analysis Kit for Optimization and Tera-scale Applications. It is developed by the Sandia National Laboratories, which is a research institution of the United States Department of Energy with almost 10,000 employees. The framework is described as a "Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis" by its developers. In other words, it is a production tool for engineering design and analysis activities and a research tool for the development of new algorithms in optimization, UQ and related areas. In addition, it can serve as a rapid prototyping tool for algorithm development. The goal of this framework is providing engineers with an effective tool for analyzing and improving existent algorithms using simulation-based models. It was originally planned to be an optimization tool only, but the analytical aspects have recently moved more into focus. It offers a variety of methods for analyzing and optimizing simulation code and contains algorithms for

- optimization with gradient and non-gradient-based methods,
- uncertainty quantification with sampling, reliability, stochastic expansion and epistemic methods,
- parameter estimation with nonlinear least squares methods and
- sensitivity/variance analysis with design of experiments and parameter study methods.

These methods can be used on their own or as components within advanced strategies, such as hybrid and surrogate-based optimization, mixed integer nonlinear programming and optimization under uncertainty.

In the following, we will provide a short instruction on how to install and run DAKOTA. Additionally, we will describe how communicating with the framework works in general. Despite the huge amount of possibilities that DAKOTA offers, in this thesis we only focus on the UQ aspect, especially on the Monte Carlo and stochastic collocation methods previously explained. More detailed information about the framework and the use of other possible methods can be found in [Ada+14a], [Ada+14b], [Ada+14c] and [Ada+14d].

3.4.1 Installation

In this section we will describe how to build version 6.0 of the DAKOTA framework from source files and run a quick test, afterwards. Note that this installation was performed on Ubuntu 12.04 in a bash shell. It is not guaranteed that it works for other Ubuntu versions and Linux distributions in exactly the same way.

Required Software At first, it has to be assured that all prerequisites are installed. DAKOTA typically requires the following tools, compilers and libraries[3]:

- C, C++, Fortran77 and Fortran90 compilers
  - Installation command:

    ```
    $ sudo apt-get install g++ gfortran
    ```

- CMake (2.8.9 or newer)
  - Note: Ensure that it is added to your `$PATH` variable
- Boost (1.49 or newer), including lib components: system, signals, regex and filesystem
- Linear algebra/math libraries, e.g. BLAS and LAPACK
  - Installation command:

    ```
    $ sudo apt-get install liblapack-dev libblas-dev
    ```

Building DAKOTA from Source After installing the requirements mentioned above the system should be prepared for compiling DAKOTA by performing the following steps in the given order:

1. Download the DAKOTA source archive file:

[3] `https://software.sandia.gov/trac/dakota/wiki/PrerequisitesForCompiling`

```
$ wget http://dakota.sandia.gov/distributions/dakota/6.0/dakota-6.0-\
public.src.tar.gz
```

- Note: If you want to install a different version of DAKOTA, you have to adjust the command accordingly.

2. Extract the archive file:

```
$ tar -xzvf dakota-6.0-public.src.tar.gz
```

- Note: A new directory called "dakota-6.0.0.src" will be created in the current working directory. In the instructions below, `$DAK_SRC` refers to this directory.

3. Create a separate build directory, e.g. `$DAK_SRC/build`, for CMake to configure and build DAKOTA:

```
$ mkdir $DAK_SRC/build
```

- Note: In the instructions below, `$DAK_BUILD` refers to this directory.

4. Make a copy of the template `BuildDakotaTemplate.cmake` to customize a CMake DAKOTA build for your platform:

```
$ cp $DAK_SRC/cmake/BuildDakotaTemplate.cmake \
$DAK_SRC/cmake/BuildDakotaCustom.cmake
```

- Note: Keep the template file in the `$DAK_SRC/cmake` directory, so it can be used for subsequent DAKOTA CMake builds.

5. Update the `$DAK_SRC/cmake/BuildDakotaCustom.cmake` file to reflect your platform configuration. Instructions on how to do this are provided in this file.

- Note: For our installation we configured the path for the boost library (`BOOST_ROOT`) and the installation path (`CMAKE_INSTALL_PREFIX`). In the instructions below, `$DAK_INSTALL` refers to this installation path specification.

6. Configure DAKOTA:

```
$ cd $DAK_BUILD
$ cmake -C $DAK_SRC/cmake/BuildDakotaCustom.cmake $DAK_SRC
```

7. Build DAKOTA:

```
$ make
```

8. Install DAKOTA:

```
$ make install
```

9. Set the path and library path variables:

```
$ export PATH=$DAK_INSTALL/bin:$DAK_INSTALL/test:$PATH
$ export LD_LIBRARY_PATH=$DAK_INSTALL/lib:$DAK_INSTALL/bin:$LD_LIBRARY_PATH
```

You can check if DAKOTA has been successfully installed and linked by executing the command

```
$ dakota -v
```

It should display DAKOTA version information like this:

```
Dakota version 6.0 released May 15 2014.
Subversion revision 2677 built Oct 1 2014 22:53:41.
```

Running a first DAKOTA Example For running a simple DAKOTA study on Rosenbrock's function (see [Ros60]), you can do the following:

1. Create a working directory for DAKOTA.
2. Copy the `rosen_multidim.in` file from `$DAK_INSTALL/examples/users` into the working directory.
3. From the working directory, run:

```
$ dakota -i rosen_multidim.in -o rosen_multidim.out
```

The files `rosen_multidim.out`, `rosen_multidim.dat` and `dakota.rst` should be created in the working directory. For information about these files, see Section 4.3.3 and [Ada+14d].

There are a lot of examples, like the one above, included in the DAKOTA installation. Detailed example descriptions can be found in the DAKOTA documentation (see [Ada+14d]).

3.4.2 Interaction with the Framework

The communication between DAKOTA and a user-supplied simulation code is generally realized by reading and writing in- and output files. Figure 3.1 illustrates the very simplified "black-box" interface.

In order to start a DAKOTA run you have to provide a suitable input file first. This DAKOTA input file contains all necessary information about the study you want to perform. There are six different specification blocks, namely `environment`, `method`, `model`, `variables`, `interface` and `responses`, that may occur in such a file. The order of these sections is not relevant. Every valid input file consists of at least one `variables`, `interface`, `responses` and `method` block. Additionally, not more than one `environment` specification should appear. Figure 3.2 shows the most common relationships between the six keyword blocks. The general information that each block contains are defined as follows:

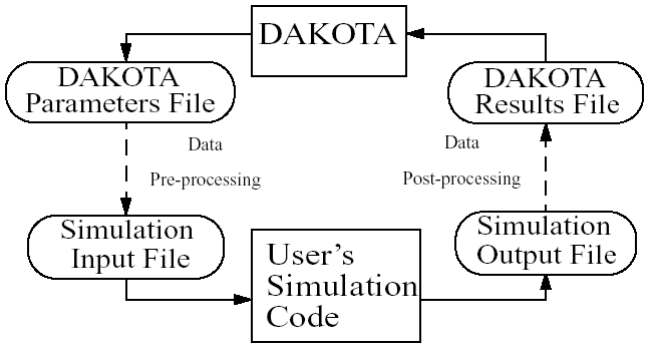


Figure 3.1: DAKOTA "black-box" interface between the framework and an user-supplied simulation code that is based on reading and writing files. The dotted lines indicate the transfer of in- and output data that has to be managed by the user. (Source: [Ada+14d])

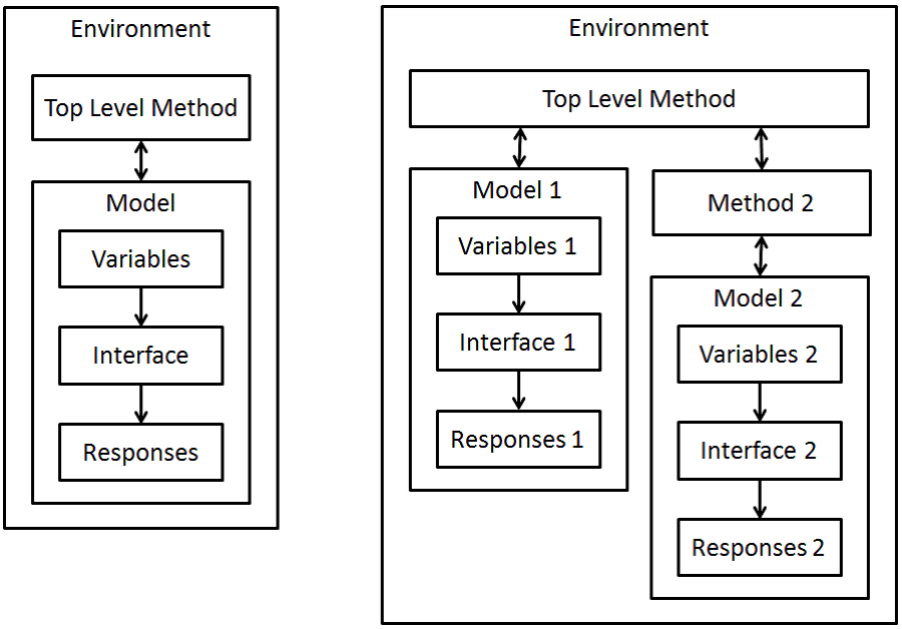


Figure 3.2: Relationship between the specification blocks of DAKOTA input files for a single model (left) and a more advanced, multi-method model (right). (Source: [Ada+14d])

environment: Used to specify general/high-level DAKOTA settings and to identify the top-level method that is used for the study.

method: Declares which method will be used by DAKOTA, e.g. parameter study, optimization, data sampling, etc.

model: Defines how variables are mapped into a set of responses. The default value for

this specification is 'single', so it is not necessary to specify this block, if none of the advanced mappings between variables and responses (surrogates or nested iterations) are desired.

variables: Describes the type (continuous vs. discrete, design, uncertain or state variables) and characteristics (name, mean, lower and upper bound, etc.) of the input parameters that are used in the study.

interface: Provides information about the mapping of parameters and responses, as well as details on the information transfer between Dakota and the simulation code.

responses: Specifies the type of data that the interface will return to DAKOTA.

Figure 3.3 contains the sample input file, that was used for the first DAKOTA study on Rosenbrock's function (see [Ros60]) above. Note that this input file uses a function as the simulation code, which is already compiled into the DAKOTA framework. Therefore, no external executable program is necessary for this study. Usually, an external simulation code is used. In this case, the executable, that DAKOTA should run during the study, has to be located together with the input file in the current working directory as well. The input files that were used for our UQ analysis and concrete descriptions of their content will be given in Section 4.3.

With a valid input file, e.g. called "dakota.in", located in the current working directory, the corresponding DAKOTA study can be executed with the command

```
$ dakota -i dakota.in
```

During execution DAKOTA produces a file with input parameters for the simulation code at the beginning of every iteration. It then runs the user-supplied code and waits for a results file. After the simulation has finished, its output information, e.g. quantities of interest, should be written to a results file. DAKOTA evaluates the content and consequently starts the next iteration by providing the next input data to the simulation code. In Figure 3.4 you can see a sample parameter and results file. If not specified else-wise, these are temporary files that are only needed for the data transfer while the study is in progress. The filenames of the files produced by DAKOTA and the simulation code during execution, as well as the number of iterations and similar information, have to be specified in the DAKOTA input file described above. Examples and information on the concrete specification can be found in Section 4.3.2 and in [Ada+14d] and [Ada+14b].

Besides the temporary results, DAKOTA produces "real" output, such as a copy of the input file, in- and output parameters of every function evaluation, mean, standard deviation, etc., that can be used for analysis purposes. The framework prints this information to the standard output (stdout). The content of the output depends on the used method and the specification of the input file. For documentation purposes it is recommended to let DAKOTA save the output to a separate file. E.g. if you run the command

```
# Dakota Input File: rosen_multidim.in
# Usage:
#   dakota -i rosen_multidim.in -o rosen_multidim.out > rosen_multidim.stdout
environment
  graphics
  tabular_graphics_data
    tabular_graphics_file = 'rosen_multidim.dat'

method
  multidim_parameter_study
    partitions = 8 8

model
  single

variables
  continuous_design = 2
    lower_bounds     -2.0     -2.0
    upper_bounds      2.0      2.0
    descriptors       'x1'     "x2"

interface
  analysis_driver = 'rosenbrock'
    direct

responses
  response_functions = 1
  no_gradients
  no_hessians
```

Figure 3.3: Sample DAKOTA input file for a two-dimensional parameter study on Rosenbrock's function. (Source: [Ada+14d])

```
                     1 variables
   1.777729657012969e+00 uin
                     1 functions
                     1 ASV_1:response_fn_1
                     1 derivative_variables
                     1 DVV_1:uin
                     0 analysis_components
                     1 eval_id
```

```
3.2032
```

Figure 3.4: Sample parameter file (left) that is produced by DAKOTA and contains input information for the simulation code and result file (right) that should be generated and filled with relevant simulation output data by the user.

```
$ dakota -i dakota.in -o dakota.out
```

a file called "dakota.out" will be created in the current working directory and the output will be written to this file, instead of being printed to the standard output.

The described interface is easy to understand and use and does not require the user to be familiar with the underlying software implementation itself. E.g., if you want

to change the analyzing method, just a few keywords of the DAKOTA input file have to be modified. In addition, you do not have to provide much information about the simulation code as well. The only knowledge necessary for the framework is the name of the executable program that should be run during the study.

Often, it is not possible or desired to modify the simulation code to read and write the corresponding in- and output files. Hence, it is required to provide a wrapper, e.g. a shell or python script, that takes care of the communication between the code and DAKOTA in every iteration.

Further descriptions and examples of such a wrapper script and the usage of the UQ methods mentioned before can be found in Section 4.3. For detailed information about input and output, possible error messages, usage of other methods, the implementation of the underlying algorithms and further DAKOTA specifications, see [Ada+14a], [Ada+14b], [Ada+14c] and [Ada+14d].

4 Benchmark-Suite

In this chapter we will describe our tool chain in detail. At first we will focus on the CFD lab code that we introduced in Section 2.3. We will start off by specifying the two scenarios for which we ran the UQ studies. Afterwards, there will be an overview on the major changes on the simulation code that were introduced to prepare the code as necessary. The second half of this chapter consists of an explanation on how to combine the lab code with DAKOTA.

4.1 Benchmark Scenarios

In this thesis we focus on two typical flow channel scenarios, the backward facing step and the channel scenario with a circular obstacle. Note that in both cases only the two-dimensional approach is considered. Further, we used relatively coarse grid resolutions for all of the performed studies. The reason for that is, that we are rather interested in the functionality of our tool chain, than in the exact resulting values.

4.1.1 Backward Facing Step

The backward facing step is an interesting channel scenario for which a lot of experimental and numerical data is available.

In this scenario the inflow height of the channel is smaller than the height of the outflow. At a certain distance from the inflow, the channel instantly expands to its full height, which forms a step in the lower left corner. From that step to the end of the channel, the height remains constantly expanded. When a fluid flows from the left to the right and passes the step, the stream will slowly "sink" to the lower boundary of the channel. This causes a vortex to arise located at the bottom directly behind the step. Figure 4.1 illustrates the scenario with the geometry configuration that was used for our studies. We specified $L = 1000$ as the channel length and $H = 74$ as the channel height. The dimensions of the step were defined as $l = 0.074 \cdot 1000 = 74$ in x direction and $s = 0.5 \cdot 74 = 37$ in y direction. In order to compare our results in the end, we used the geometry specifications from [Nec05] scaled with a factor of 1000. Since the proportions of the values are the same, the scaling should not change the resulting values. [Nec05] provides an appropriate reference, since it uses a similar discretization of the Navier-Stokes-Equations and the utilized grid resolutions are not very high, either. To the boundaries at the top and bottom we applied a no-slip condition, as well as an inlet condition to the inflow on the left and an outlet condition to the outflow on the right. The mean integral value u_{in} of the parabolic inflow profile was the uncertain input

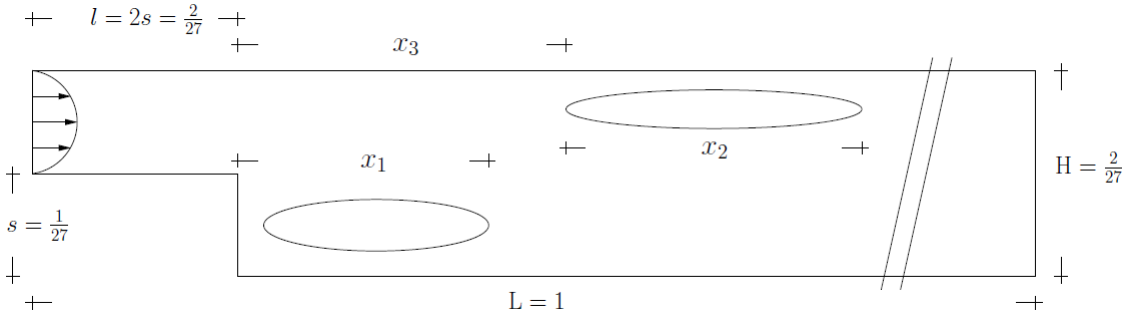


Figure 4.1: Geometry configuration for the backward facing step scenario. (Source: [Nec05])

parameter in our UQ analysis. We will further specify the uncertainty in Section 4.3.2. As values for the Reynolds number we defined $Re = 100$ and 250. For each Reynolds number we used two different grid resolutions, 135×10 and 243×18, resulting in four different setups for this scenario. In Figure 4.2 you can see the configuration file that was used for the studies on the backward facing step with $Re = 100$ and a grid resolution of 135×10. For the scenario with a Reynolds number of 250, the $Re =$ "100" specification was changed to $Re =$ "250". Furthermore, the finer grid resolution was achieved by assigning a value of 243 to "sizeX" and 18 to "sizeY".

The quantity that we are interested in, called the quantity of interest (QoI), is the length x_1 of the first vortex located at the bottom boundary behind the step (see Figure 4.1) relative to the step height s. This QoI is also called reattachment length. A short description of the implementation and calculation of the reattachment length will be given in Section 4.2.3. The values of the parameters x_2 and x_3, that can be seen in the image above, are not of interest and, therefore, will be ignored in the following.

4.1.2 Channel with Circular Obstacle

The channel with circular obstacle is another common flow channel scenario and a classic benchmark problem of an incompressible two-dimensional flow. In this scenario there is a cylinder located in the middle of the channel. Differing from the backward facing step, the height of the channel remains the same from the beginning to the end. In Figure 4.3 you can see our parameter configuration for this flow scenario.

To compare the resulting values, we copied the configuration for this scenario from [Men14]. We used $L = 2.2$ as the length and $H = 0.41$ as the height of the channel. Additionally, some obstacle parameters were needed for this scenario: we located the cylinder with a radius of $r = 0.05$ at $x = 0.2$ and $y = 0.2$. The Reynolds number was initialized with $Re = 20$ and $Re = 100$. This time, we only used one grid resolution, a 220×41 grid, for both Reynolds numbers. Thus, we obtained two different setups for the obstacle channel. The configuration file that we used for analyzing the obstacle channel scenario with $Re = 20$ can be seen in Figure 4.4. For the setup with a Reynolds number

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <flow Re="100" />
    <simulation finalTime="1000" epsilon="0.000001">
        <type>dns</type>
        <scenario>channel</scenario>
    </simulation>
    <backwardFacingStep xRatio="0.074" yRatio="0.5" />
    <timestep dt="0.5" tau="0.5" />
    <solver gamma="0.5" />
    <geometry
      dim="2"
      dx="0" dy="0" dz="0"
      sizeX="135" sizeY="10" sizeZ="0"
      lengthX="1000" lengthY="74" lengthZ="1" >
      <mesh>uniform</mesh>
        <!--<mesh>stretched</mesh>-->
    </geometry>
    <environment gx="0" gy="0" gz="0" />
    <walls>
        <left>
            <vector x="1.0" y="0" z="0" />
        </left>
        <right>
            <vector x="0" y="0" z="0" />
        </right>
        <top>
            <vector x="0" y="0" z="0" />
        </top>
        <bottom>
            <vector x="0" y="0" z="0" />
        </bottom>
        <front>
            <vector x="0" y="0" z="0" />
        </front>
        <back>
            <vector x="0" y="0" z="0" />
        </back>
    </walls>
    <vtk interval="0.1">channel_result</vtk>
    <stdOut interval="0.01" />
    <parallel numProcessorsX="1" numProcessorsY="1" numProcessorsZ="1" />
</configuration>
```

Figure 4.2: Configuration file for the backward facing step scenario with $Re = 100$.

of 100, the inflow velocity u_{in} was changed to 1.0. Note that the value 1000, specified as Re in the configuration file of the obstacle channel scenario, does not represent the actual Reynolds number. Further explanations will be given in Section 5.2. The applied boundary conditions were the same as for the backward facing step. The uncertain variable for the obstacle channel scenario was again the inflow velocity u_{in}.

Our quantities of interest in this scenario are the coefficients c_d and c_l of the drag and lift forces that act on the obstacle in the stream. These are defined as follows:

$$c_d = \frac{2F_d}{\rho u_\infty^2 L_\infty} \quad , \qquad c_l = \frac{2F_l}{\rho u_\infty^2 L_\infty} \quad , \tag{4.1}$$

where F_d and F_l are the forces of drag and lift, ρ is the density of the fluid and u_∞ and

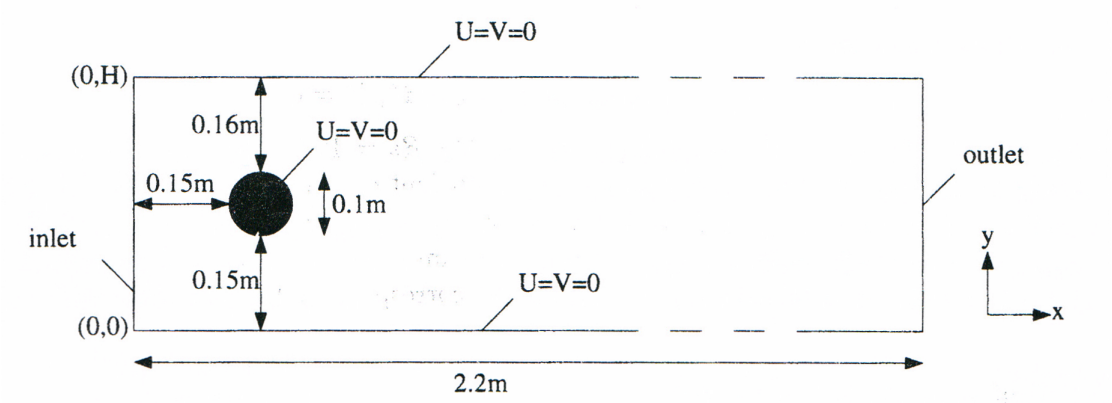


Figure 4.3: Geometry configuration for the obstacle channel scenario. (Source: [Men14])

L_∞ denote the reference velocity and length. In this scenario, the reference quantities are defined as inflow velocity u_{in} and diameter $d = 2 \cdot r$ of the obstacle. The implementation of the round obstacle and the calculation of the drag and lift coefficients are described in Section 4.2.4. The expansion of the obstacle channel to a three-dimensional scenario can be seen in [Sch+96].

4.2 Preparation of the CFD Lab Code

Due to the fact that the original code from the CFD lab did not provide all the functionalities that we needed for the simulation runs, some modifications had to be applied. All of the implementations described in the following are written in C++, as is most of the simulation code. The description of the code changes will mention the terms "iterator" and "stencil" at some points. To avoid ambiguity, we will introduce the principle it is referring to, before beginning with the description of the implementations itself. As described in Section 2.3.1, the flow field is represented as a staggered grid. To access the values of the grid cells the code makes use of iterators and stencils. An iterator iterates over several particular cells and applies a stencil to each. The stencil can access the information of the cells and process it. Which cells are visited by the iterator and what exactly is done to each of them is specified in the iterator and stencil implementation.

4.2.1 Structure of the Output

At first we changed the structure of the output that the code produces after every execution. The original code simply saved the vtk files mentioned in Section 2.3.3 into the root directory of the program. Thus, it was overwritten after the next execution and, therefore, not possible to review results of prior runs if the user did not save it manually. To avoid this loss of information, we added a directory, called "results", with the two subdirectories "bfstep" and "channel", one for the backward facing step and the other

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <flow Re="1000" />
    <simulation finalTime="15" epsilon="0.000001">
        <type>dns</type>
        <scenario>round-obstacle-channel</scenario>
    </simulation>
    <backwardFacingStep xRatio="0.0" yRatio="0.0" />
    <obstacle xOffset="0.2" yOffste="0.2" radius="0.05" />
    <timestep dt="0.001" tau="0.5" />
    <solver gamma="0.5" />
    <geometry
      dim="2"
      dx="0" dy="0" dz="0"
      sizeX="220" sizeY="41" sizeZ="0"
      lengthX="2.2" lengthY="0.41" lengthZ="1" >
      <mesh>uniform</mesh>
        <!--<mesh>stretched</mesh>-->
    </geometry>
    <environment gx="0" gy="0" gz="0" />
    <walls>
        <left>
            <vector x="0.2" y="0" z="0" />
        </left>
        <right>
            <vector x="0" y="0" z="0" />
        </right>
        <top>
            <vector x="0" y="0" z="0" />
        </top>
        <bottom>
            <vector x="0" y="0" z="0" />
        </bottom>
        <front>
            <vector x="0" y="0" z="0" />
        </front>
        <back>
            <vector x="0" y="0" z="0" />
        </back>
    </walls>
    <vtk interval="0.1">channel_result</vtk>
    <stdOut interval="0.01" />
    <parallel numProcessorsX="1" numProcessorsY="1" numProcessorsZ="1" />
</configuration>
```

Figure 4.4: Configuration file for the obstacle channel scenario (round-obstacle-channel) with an inflow velocity $u_{in} = 0,2$ to achieve a Reynolds number of 20.

for the channel with obstacle scenario. By executing the code, it now creates a directory - named with the system time - in the corresponding scenario directory. Alternatively, the name of this newly created folder can be determined, by passing the desired name as an additional argument to the code. For example, if you run the program with

```
$ ./ns conf_channel.xml my_output_directory_name
```

it will be executed with `conf_channel.xml` as the configuration file and afterwards, an output directory called "my_output_directory_name" will be created in the corresponding scenario directory. The vtk files that have been created during the simulation, as well as a copy of the used configuration file and the according bash output are stored

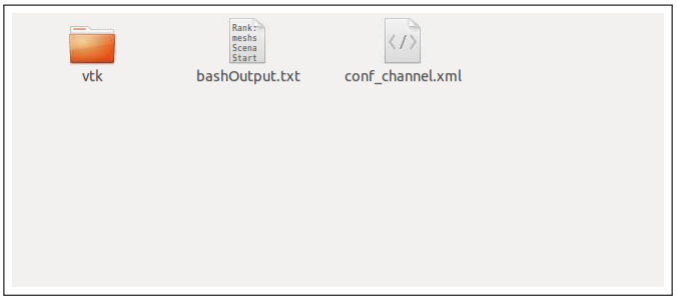


Figure 4.5: Content of the output folder `my_output_directory_name` that is created after the simulation code execution has finished. It consists of the copied configuration file `conf_channel.xml`, a text file `bashOutput.txt` with the corresponding bash or standard output and a `vtk` directory with the vtk files.

in this folder. Its content can be seen in Figure 4.5. This modification of the output structure was necessary for the documentation and evaluation of multiple simulation runs, which plays a major role in this thesis.

4.2.2 Channel with Circular Obstacle

The next step was the implementation of the channel scenario with a circular obstacle. The original code supported the backward facing step and an empty channel scenario, but not the obstacle channel setting described earlier. To adjust that, we modified the empty channel with an additional stencil which adds a round obstacle. The stencil calculates the distance from the center of the obstacle for every cell. If the distance is smaller or equal to the desired radius, the cell is flagged as an obstacle cell; otherwise it is not. The parameters of the obstacle can be specified via the configuration file. The corresponding keywords and their meanings are:

xOffset: x position of the center (absolute)

xOffsetRatio: x position of the center (relative to the width of the flow field)

yOffset: y position of the center (absolute)

yOffsetRatio: y position of the center (relative to the height of the flow field)

radius: radius r of the circular obstacle

It is only possible to define either the absolute or the relative offset for each dimension. As mentioned in the scenario description, Figure 4.4 shows a configuration file for one of the conducted circular obstacle setups. In this case, the obstacle center is located at $(x, y) = (0.2, 0.2)$ and the radius is defined as 0.05.

4.2.3 Reattachment Length

In order to use the quantities of interest that were mentioned in the scenario description above, we had to implement their calculation and output first. We started off with the reattachment length, which is the quantity of interest for the backward facing step scenario. As declared before, the reattachment length is the length of the vortex located directly behind the backward facing step. This vortex pushes the fluid at the bottom of the flow field towards the step (see Figure 4.1). Therefore, the velocity x-values of the cells of the vortex in the row at the bottom boundary of the flow field must be negative. This means that we have to find the first cell in the lowermost grid row with positive velocity in x direction to calculate the reattachment length.

For this calculation, a new iterator that iterates over all cells of the first flow field row was added and a corresponding stencil was applied to it. When this stencil detects the first positive x-velocity, the reattachment length is calculated from the position of the detection, the cell width and the width of the backward facing step. Given that the cell with positive x-velocity is actually not part of the vortex, it is excluded from the reattachment length. After the calculation, the value for the reattachment length is divided by the step height s, to obtain a dimensionless quantity, and saved in a .txt file in the corresponding output directory.

4.2.4 Drag and Lift Coefficients

For the calculation of the drag and lift coefficients c_d and c_l, we also added a new iterator and stencil. The iterator iterates over all cells that are directly adjacent to cells of the obstacle but not obstacle cells themselves. The stencil, that is applied to these "edge cells", detects the location of the current cell relative to the obstacle and uses its pressure value to manipulate the drag and lift coefficients as follows:

- cell on the left side of the obstacle $\rightarrow$ pressure is added to c_d
- cell on the right side of the obstacle $\rightarrow$ pressure is subtracted from c_d
- cell on the bottom of the obstacle $\rightarrow$ pressure is added to c_l
- cell on the top of the obstacle $\rightarrow$ pressure is subtracted from c_l

With this procedure, we get the pressure part of the stress tensor of the drag and lift forces F_d and F_l, which is stored in the variables c_d and c_l after the iteration has finished. Since the partial pressure calculated above is the dominant part of the forces, we can simplify the force calculations by ignoring the viscous parts. To obtain the desired drag and lift coefficients from the approximated forces, we need to norm the forces by multiplying them with the norm factor $\frac{2}{\rho u_\infty^2 L_\infty}$ (see (4.1)). Similar to the reattachment length for the backward facing step, the quantities of interest of the obstacle channel scenario are written to a .txt file in the corresponding output directory.

4.2.5 Termination Condition

The last major change was the implementation of a termination condition. Its goal was that the simulation should terminate, when the flow field reaches a stationary state. We realized this criterion by calculating the change in velocity of the whole flow field in every time step. For this calculation, we used the discrete L^2 norm

$$||u^{(n+1)} - u^{(n)}|| = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(u_i^{(n+1)} - u_i^{(n)})^2} \ , \tag{4.2}$$

where N is the number of cells of the whole flow field and $u^{(n+1)}$ and $u^{(n)}$ are vectors that consist of N entries. Every entry i stands for the length of the velocity vector of cell i in iteration $(n+1)$ and (n):

$$\begin{aligned} u_i^{(n+1)} &= ||u_i^{(n+1)}|| = \sqrt{\left(u_i^{(n+1)}\right)_x^2 + \left(u_i^{(n+1)}\right)_y^2} \ , \\ u_i^{(n)} &= ||u_i^{(n)}|| = \sqrt{\left(u_i^{(n)}\right)_x^2 + \left(u_i^{(n)}\right)_y^2} \ . \end{aligned} \tag{4.3}$$

Equation (4.2) calculates the absolute velocity difference of the flow field from time step (n) to $(n+1)$. Due to the fact, that we do not care about the dimensions of the velocity values, we calculate the relative change

$$\frac{||u^{(n+1)} - u^{(n)}||}{||u^{(n)}||} \ . \tag{4.4}$$

The simulation terminates in iteration $(n+1)$, if the relative velocity difference (4.4) is smaller than ϵ:

$$\frac{||u^{(n+1)} - u^{(n)}||}{||u^{(n)}||} < \epsilon \ , \tag{4.5}$$

where ϵ can be configured in the simulation block of the configuration file with the keyword `epsilon`.

This modification of the original lab code was not actually required for the studies to work. However, it made working with the simulation code and analyzing its results a lot easier.

4.3 Coupling of the CFD Lab Code with DAKOTA

In this section, we will describe how to connect DAKOTA to the simulation code. The general principle of the communication between user-supplied code and DAKOTA can be found in Section 3.4.2 and in the [Ada+14d].

```
#!/bin/sh
# Wrapper script for the backward facing step scenario that is used to transfer
# in- and output data between DAKOTA and the simulation code

# --------------
# PRE-PROCESSING
# --------------

# Incorporate the parameters from DAKOTA into the template, writing conf_channel.xml
dprepro params.in conf_channel_template.xml conf_channel.xml


# --------
# ANALYSIS
# --------

# change the directory to the simulation code, execute the simulation
# and switch back to the dakota analysis directory
cd $SIMULATION_PATH
./ns $DAK_ANALYSIS_PATH/conf_channel.xml default
cd -


# ---------------
# POST-PROCESSING
# ---------------

# extract function value from the simulation output
grep -i 'Dimensionless reattachment length' $SIMULATION_PATH/results/default/bashOutput.txt \
| cut -c 35- >> results.tmp

mv results.tmp results.out
```

Figure 4.6: Wrapper script for the backward facing step scenarios that takes care of the data transfer between DAKOTA and the simulation code.

4.3.1 Wrapper Scripts

In order to connect our code to DAKOTA we need to transfer the (temporary) output from DAKOTA to the input of the simulation code and vice versa in every iteration of the DAKOTA run. Our simulation code is not able to read and interpret the produced files of DAKOTA and, additionally, does not provide the QoI in the correct format. Since we do not want to modify the implementation of our code to do so, we need to attach a wrapper script in between of the two applications. In this case, we developed two simple shell scripts to manage the data transfer, one for the backward facing step and one for the obstacle channel. In Figure 4.6, the script for the backward facing step analysis can be seen, for which we will shortly explain each line of code:

1. **Pre-Processing**
   - Command:

     `$ dprepro params.in conf_channel_template.xml conf_channel.xml`

     – Description: The `dprepro` command incorporates relevant parameter values from the file `params.in` into the configuration file template `conf_channel_template.xml`.

Thus, it creates a new configuration `conf_channel.xml` with the inserted values. The template file itself is not modified at all and will be used as a template in every iteration. (See Figure 4.7 for an illustration of a template configuration file)

- Note: The file called `dprepro` has to be located in the current working directory to make this command work. It can be found in the examples included in DAKOTA, more precisely in the `$DAK_INSTALL/examples/script_interfaces/generic` directory. Alternatively, one can use the `aprepro` command to insert the parameters into the configuration file.

2. **Analysis**
   - Command:

     ```
     $ cd $SIMULATION_PATH
     ```

     - Description: This command switches the current working directory to the directory of the simulation code.
     - Note: `$SIMULATION_PATH` refers to the directory of the simulation code installation.
   - Command:

     ```
     $ ./ns $DAK_ANALYSIS_PATH/conf_channel.xml default
     ```

     - Description: This is the actual execution command for the simulation code. The file `ns` is the executable program of the simulation code. The second argument is the path to the configuration file that was created in the pre-processing phase by the `dprepro` command above. The third argument is the name of the directory where the output data of one simulation run is stored. In this case, it is saved in `$SIMULATION_PATH/results/default` and will be overwritten in every iteration.
     - Note: `$DAK_ANALYSIS_PATH` refers to the DAKOTA working directory where the analysis was executed from.
   - Command:

     ```
     $ cd -
     ```

     - Description: With this command you can switch back to the previous directory. Here, the working directory is changed to `$DAK_ANALYSIS_PATH`.
3. **Post-Processing**
   - Command:

     ```
     $ grep -i 'Dimensionless reattachment length' \
     $SIMULATION_PATH/results/default/bashOutput.txt \
     | cut -c 35- >> results.tmp
     ```

  - Description: This command filters our quantity of interest, the reattachment length, from the simulation output, the `bashOutput.txt` file in the output directory of the simulation code specified in the execution command above, and writes it to a temporary file `results.tmp`.

- Command:

```
$ mv results.tmp results.out
```

  - Description: This command changes the name of the temporary file to `results.out`, which is the name of the results file that DAKOTA expects. (See specification of the DAKOTA input file in Section 4.3.2)
  - Note: In this case, the creation of the temporary file would not be necessary, since there is only one output parsing line. If you have more than one QoI that needs to be copied into the results file, it is recommended to create this temporary file first, insert all the important data separately and change the name afterwards. With this method you can make sure that the `results.out` file actually contains all of the relevant values.

The only change necessary for the obstacle channel is located in the post-processing section. Instead of one, the script has to copy two quantities of interest and, therefore, read two lines of the simulation output. The post-processing block for this scenario looks like the following:

```
$ grep -i 'Drag coefficient' \
$SIMULATION_PATH/results/default/bashOutput.txt \
| cut -c 18- >> results.tmp
$ grep -i 'Lift coefficient' \
$SIMULATION_PATH/results/default/bashOutput.txt \
| cut -c 18- >> results.tmp
$ mv results.tmp results.out
```

Regard that the script has to be located in the working directory from which you run DAKOTA, if not specified else-wise.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <flow Re="1000" />
    <simulation finalTime="15" epsilon="0.000001">
        <type>dns</type>
        <scenario>round-obstacle-channel</scenario>
    </simulation>
    <backwardFacingStep xRatio="0.0" yRatio="0.0" />
    <obstacle xOffset="0.2" yOffste="0.2" radius="0.05" />
    <timestep dt="0.001" tau="0.5" />
    <solver gamma="0.5" />
    <geometry
      dim="2"
      dx="0" dy="0" dz="0"
      sizeX="220" sizeY="41" sizeZ="0"
      lengthX="2.2" lengthY="0.41" lengthZ="1" >
      <mesh>uniform</mesh>
        <!--<mesh>stretched</mesh>-->
    </geometry>
    <environment gx="0" gy="0" gz="0" />
    <walls>
        <left>
            <vector x="{uin}" y="0" z="0" />
        </left>
        <right>
            <vector x="0" y="0" z="0" />
        </right>
        <top>
            <vector x="0" y="0" z="0" />
        </top>
        <bottom>
            <vector x="0" y="0" z="0" />
        </bottom>
        <front>
            <vector x="0" y="0" z="0" />
        </front>
        <back>
            <vector x="0" y="0" z="0" />
        </back>
    </walls>
    <vtk interval="0.1">channel_result</vtk>
    <stdOut interval="0.01" />
    <parallel numProcessorsX="1" numProcessorsY="1" numProcessorsZ="1" />
</configuration>
```

Figure 4.7: Template for the backward facing step configuration file. The wrapper script substitutes the "{uin}" entry with the corresponding value of the inflow velocity u_{in} from the parameter file that is generated by DAKOTA.

```
#  DAKOTA INPUT FILE - dakota_test.in

environment,
  tabular_graphics_data
    tabular_graphics_file = 'dakota_test.dat'

method,
  sampling
    sample_type random
    samples = 100
    seed = 27

model,
  single

variables,
  normal_uncertain = 1
    means           1.0
    std_deviations  0.1
    descriptors     'uin'

interface,
  fork
    analysis_driver = 'simulator_script'
    parameters_file = 'params.in'
    results_file    = 'results.out'

responses,
  response_function = 1
  no_gradients
  no_hessians
```

Figure 4.8: DAKOTA input file for the Monte Carlo runs of the backward facing step.

4.3.2 DAKOTA Input Files

Besides the wrapper scripts taking care of the communication, we also need a valid input file for each UQ method, in order to run the DAKOTA studies. In the following we will describe each block of the input file (see Figure 4.8), that was used for the Monte Carlo studies on the backward facing step, in detail.

- **environment**
  - `tabular_graphics_data`: With this keyword DAKOTA creates a tabular data output file that contains input variables and corresponding responses for every function evaluation. It can be used for plotting/visualization purposes.
    - `tabular_graphics_file = 'dakota_test.dat'`: Specifies "dakota_test.dat"

as the name for the tabular data output file.

- **method**
  - `sampling`: Defines UQ based on random sampling as the method to use.
    - `sample_type = random`: Chooses "pure" random as the sampling method. Alternative: Latin Hypercube Sampling.
    - `samples = 100`: Sets the number of evaluations to 100.
    - `seed = 27`: Determines 27 as the seed for the random number generator. If not specified, the system time is used as the default seed.
- **model**
  - `single`: Specifies that a model with exactly one of each of the blocks `variables`, `interface` and `responses` is used. Could be omitted because the single model is specified by default.
- **variables**
  - `normal_uncertain = 1`: Means that one uncertain normal distributed input variable exists.
    - `means 1.0`: Defines 1.0 as the mean for the uncertain variable.
    - `std_deviations 0.1`: Defines 0.1 as the standard deviation for the uncertain variable. Note that 0.1 is comparatively high for the standard deviation of an uncertain input velocity. Due to the coarse resolutions that were used for the studies and the "robustness" of the reattachment length, we needed to specify such an unrealistic value. Otherwise, no noticeable fluctuations would occur in the resulting QoI.
    - `descriptors 'uin'`: Labels the uncertain variable with the name "uin".
- **interface**
  - `fork`: Means that the analysis driver is launched by using a fork command, which implies that Dakota launches a separate application analysis process.
    - `analysis_driver = 'simulator_script'`: Specifies the name of the program that DAKOTA executes in the analysis phase as "simulator_script". In this case it is the name of the wrapper script previously mentioned.
    - `parameters_file = 'params.in'`: Specifies the name of the parameter file that DAKOTA provides to the simulation code at the start of each iteration as "params.in".
    - `results_file = 'results.out'`: Specifies the name of the result file that DAKOTA expects to be generated by the user-supplied code at the end of each iteration as "results.out".
- **responses**

- `response_function = 1`: Declares that there is one response function that produces generic data with no special interpretation taken by the method in use. In this case the returned data is the reattachment length from the backward facing step.
- `no_gradients`: Indicates that gradient information is not needed in this study and, therefore, will neither be returned by the simulation code nor calculated by DAKOTA.
- `no_hessians`: Indicates that the method does not require any Hessian information.

For analyzing the circular obstacle channel scenario, you have to specify two response functions in the responses block. That is because of the two QoI, the drag and lift coefficients, instead of only one.

If you want to start an analysis using a stochastic collocation method, you have to modify the method specification. The `sampling` keyword has to be replaced by `stoch_collocation`. In our stochastic collocation studies we added the `quadrature_order` keyword to the method section, in order to use tensor-products for node selection. The number you have to assign to it specifies the number of the quadrature level. In Figure 4.9 you can see our sample input file for an obstacle channel analysis using stochastic collocation. For the studies on the obstacle channel, we used a standard deviation of 0.005, which can be considered as a realistic value, e.g. caused by measurement errors. As the input file shown in Figure 4.9 was used for the $Re = 20$ setup, the mean of the uncertain input, i.e. the inflow velocity u_{in}, is determined as 0.2. For the $Re = 100$ specification, it hast to be changed to 1.0. The standard deviation stays the same in both cases.

```
#  DAKOTA INPUT FILE - dakota_test.in

environment,
  tabular_graphics_data
    tabular_graphics_file = 'dakota_test.dat'

method,
  stoch_collocation
    quadrature_order = 7
    seed = 27

model,
  single

variables,
  normal_uncertain = 1
    means           0.2
    std_deviations  0.005
    descriptors     'uin'

interface,
  fork
    analysis_driver = 'simulator_script'
    parameters_file = 'params.in'
    results_file    = 'results.out'

responses,
  response_function = 2
  no_gradients
  no_hessians
```

Figure 4.9: DAKOTA input file for the stochastic collocation runs of the obstacle channel.

4.3.3 Running the UQ Experiments

As mentioned in Section 3.4.2, you can run DAKOTA with the input file `dakota_test.in` by executing the command

```
$ dakota -i dakota_test.in -o dakota_test.out
```

To make sure that this works properly, the input file, the wrapper script, the configuration template and the `dprepro` file have to be located in the working directory from which the above command is executed. During execution, the following files should appear in the directory:

- `dakota.rst`: A restart file that DAKOTA can use to restart an analysis, if it was interrupted. To make use of this feature, you can restart the study by executing

  ```
  dakota -i dakota\_test.in -r dakota.rst -o dakota\_test.out
  ```

  DAKOTA will skip the sample runs that were performed before the interruption and continue with the interrupted execution.

- `dakota_test.dat`: Tabular graphics file that contains input variables and responses for each function evaluation and can be used for plotting/visualization purposes.

- `dakota_test.out`: DAKOTA output file that contains overall information about the study, such as a copy of the input file, entries for every evaluation/iteration, mean, standard deviation, etc.

- `conf_channel.xml`: Configuration file that was created by the simulator script in the last iteration.

- `S7` and `LHS_*.out` files: Files that are relevant, if the Latin Hypercube Sampling method was selected as the sample type. They are created with the "pure" random sample strategy anyway, but can be ignored in this case.

Note that, with these two studies, we only make use of a small amount of the possibilities that the DAKOTA framework offers. Furthermore, there are even a lot of possible modifications, such as creating additional files, changing the simulation structure, modifying the output file, etc., that can be applied to our studies by using different or additional keyword specifications in the input file. For those, who are willing to try different methods or specifications, we recommend reading [Ada+14a], [Ada+14b], [Ada+14c] and [Ada+14d].

5 Numerical Results

In the following we will present the analysis results of the two scenarios declared in Section 4.1. For each scenario, we will first list the results for the different deterministic runs. These deterministic outcomes will be compared with the data listed in the referenced literature to validate the behavior of our simulation code. Afterwards, we will present the results of the DAKOTA runs, where we applied uncertainty to input variables, and compare their evaluation with the results of the deterministic runs.

5.1 Backward Facing Step

As mentioned in the scenario description, we studied four different setups for the backward facing step, two different grid resolutions, 135×10 and 243×18, with two different Reynolds numbers, $Re = 100$ and 250, each.

5.1.1 Deterministic Results

We first performed a deterministic run with a fixed input velocity $u_{in} = 1.0$ for each of the four setups. In figure 5.1 the last time step of the deterministic runs of each setup is visualized. Table 5.1 compares our results for the reattachment length x_1 relative to the step height s with the numerical results of the 243 grid of [Nec05].

	$Re = 100$			$Re = 250$		
	135×10	243×18	[Nec05] (243 grid)	135×10	243×18	[Nec05] (243 grid)
$\frac{x_1}{s}$	2.002	3.003	2.94	2.002	7.897	5.83

Table 5.1: Dimensionless reattachment length $\frac{x_1}{s}$ for each of the deterministic runs of the different backward facing step scenarios compared with the results of [Nec05].

It can be seen, that, especially for the Reynolds number $Re = 250$, the values do not match. This can be attributed to the fact, that we used a simplified reattachment length calculation, as it is limited to a multiple of the cell width δx. Thus, the computation of the reattachment length is not as accurate, as it is in [Nec05]. However, these small divergences are not crucial, since we do not want to use the results for precise physical evaluations. This comparison should answer the purpose of a validation of the simulation code, which, based on the comparatively small errors, can be considered as successful.

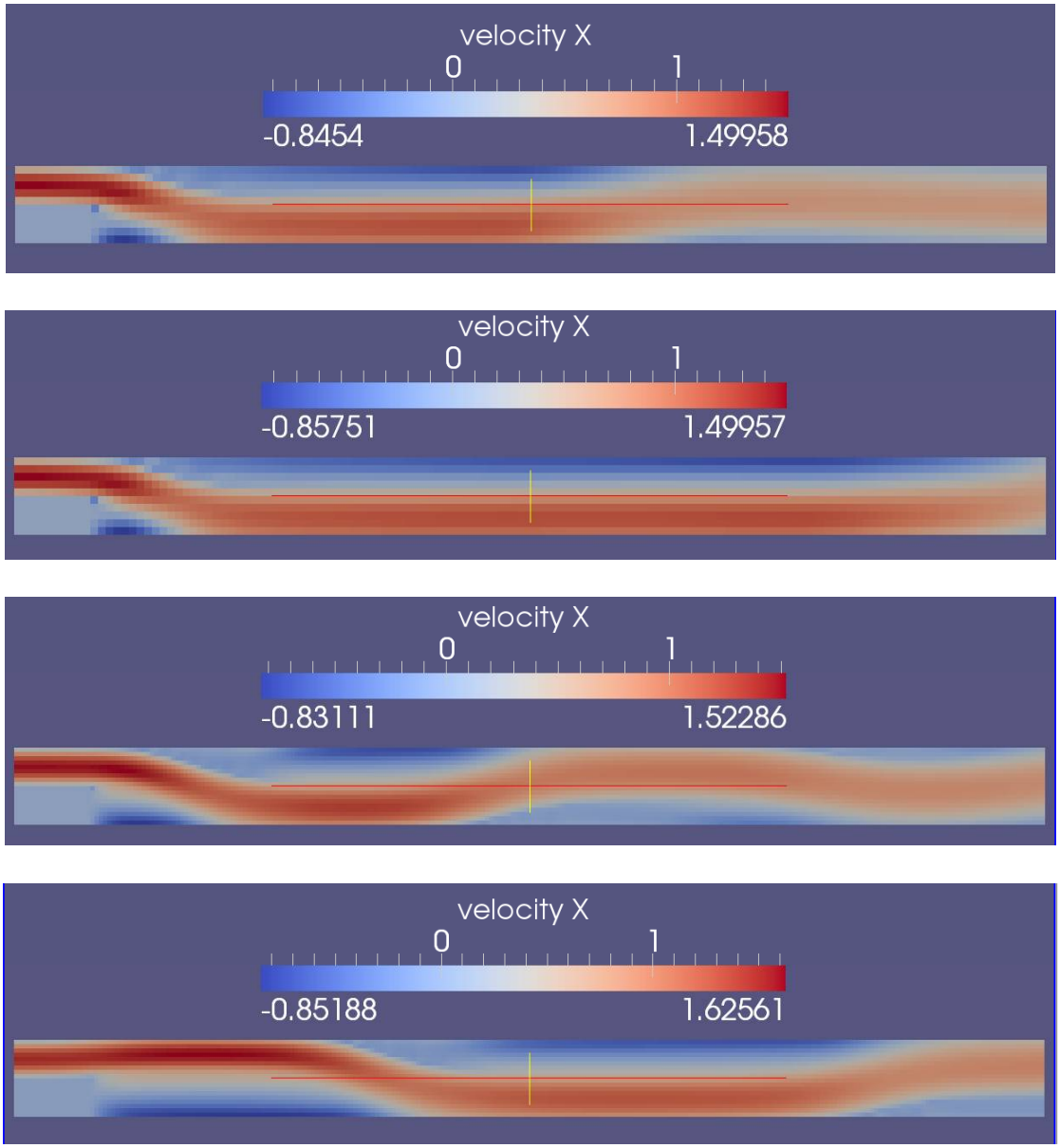


Figure 5.1: Visualizations by Paraview of the velocities in x direction of the last time step of the deterministic run for each of the four backward facing step setups. First: 135×10 grid with $Re = 100$. Second: 135×10 grid with $Re = 250$. Third: 243×18 grid with $Re = 100$. Fourth: 243×18 grid with $Re = 250$.

5.1.2 UQ Results

Given that the deterministic solver works correctly, we can move forward to the more significant DAKOTA runs. For each of the four setups mentioned before, we performed two DAKOTA studies, one with Monte Carlo sampling and one with stochastic collo-

cation. For all of the DAKOTA runs, we specified a mean $\mathbb{E}(u_{in}) = 1.0$ and a standard deviation $\sigma_{u_{in}} = 0.1$ for the uncertain inflow velocity u_{in}. Since we used $u_{\text{in}} = 1.0$ as the inflow velocity of the deterministic studies, this configuration provides data to compare with. Table 5.2 shows the mean and standard deviation of the QoI for both UQ methods for each backward facing step configuration. We specified 100 samples for MCS and a quadrature order of 7 for the SC method.

		$Re = 100$		$Re = 250$	
		135×10 grid	243×18 grid	135×10 grid	243×18 grid
MCS	$\mathbb{E}(\frac{x_1}{s})$	2.002	2.996	1.954	7.813
	$\sigma_{\frac{x_1}{s}}$	0.0	0.027	0.086	0.873
SC	$\mathbb{E}(\frac{x_1}{s})$	2.002	2.999	1.948	7.865
	$\sigma_{\frac{x_1}{s}}$	0.0	0.019	0.089	0.62

Table 5.2: Mean and standard deviation of the dimensionless reattachment length $\frac{x_1}{s}$ for each of the backward facing step DAKOTA studies.

By comparing the means with the deterministic results listed in Table 5.1, one can see, that the reattachment length is considerably robust in the face of input uncertainty, at least for the used grid resolutions. Despite adjusting the standard deviation of the uncertain input velocity to a "non-realistic" value, the result of the 135×10 grid barely varies. Furthermore, it is conspicuous that the SC analysis is almost as accurate as the MCS runs are. Although, SC needed significantly less simulation runs and, therefore, execution time than MCS did, the resulting statistical information is very similar. E.g. for the 243×18 grid with $Re = 250$, the SC analysis took about 6 hours, whereas the MCS study almost took 87 hours. However, the SC result is slightly more accurate.

Since DAKOTA provides tabular data files with in- and output data of every sample, we can plot some of the runs for a visualization of the UQ method behavior. Figure 5.2 shows the reattachment length in every iteration of the Monte Carlo and stochastic collocation methods of both 243×18 backward facing step configurations.

In the plots of the $Re = 100$ scenarios, one can see the non-linear behavior, that results from the simplification of the reattachment length calculation. Furthermore, both plots illustrate the general behavior and difference between the MCS and SC methods quite good.

5.2 Channel with Circular Obstacle

This section follows the same structure as the result section for the backward facing step. We used the same grid resolution of 220×41 cells for both Reynolds numbers, $Re = 20$ and 100. Pay attention to the fact that the value 1000 that we specified in Figure 4.4 for Re is $\frac{1}{\eta}$, where $\eta = 10^{-3}$ is the kinematic viscosity of the fluid, and not the actual Reynolds number. Since the parameter configuration of [Men14] was used, we applied

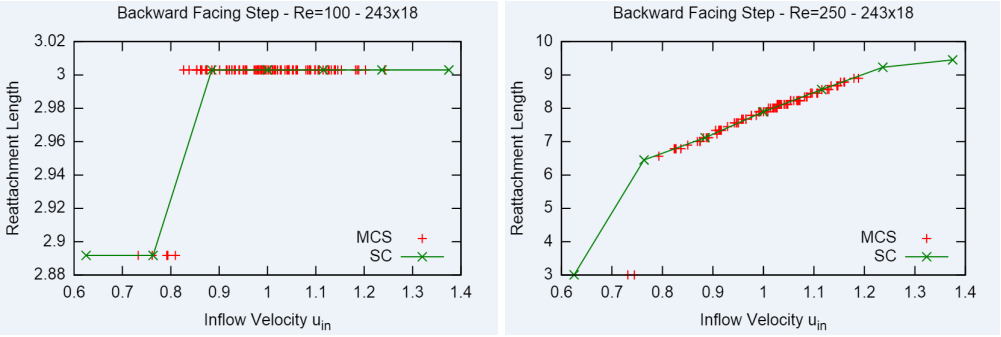


Figure 5.2: Every Sample of the MCS and SC analysis with $Re = 100$ (left) and $Re = 250$ (left).

this "trick" to avoid converting the dimensional variables into dimensionless ones and vice versa.

5.2.1 Deterministic Results

The deterministic runs were executed with an input velocity u_{in} of 0.2 and 1.0 to achieve the desired Reynolds numbers of 20 and 100. Table 5.3 presents the force coefficients resulting from the deterministic runs of the obstacle channel scenario. This time we compare it with the results of [Men14], as we used its scenario description as a guideline for our channel configuration. Due to the fact that there are only plots of c_d and c_l provided in [Men14], we had to determine the reference values graphically, which may add inaccuracy. Furthermore, we could not determine a specific value for the lift coefficient c_l of the $Re = 100$ analysis, since it oscillates around the value $c_l = 0$. The grid resolution of the reference study is the same as ours.

	$Re = 20$		$Re = 100$	
	220×41 grid	[Men14]	220×41 grid	[Men14]
c_d	4.988	4.95	4.007	~ 2.65
c_l	10^{-7}	0.022	-0.18	-

Table 5.3: Drag and lift coefficients c_d and c_l for each of the deterministic runs of the different obstacle channel scenarios compared to the results of [Men14].

Again, one can derive from this illustration, that our results do not perfectly match the reference values. Especially in the lower value range, the simulation code does not provide the appropriate accuracy, since we approximated the QoI calculation for the obstacle channel scenario as well. The impreciseness for the $Re = 100$ configuration is the result of the oscillating drag and lift coefficient. As can be seen in [Men14], the flow never reaches a steady state for this Reynolds number configuration. This unsteady behavior can also be noticed in figure 5.3, which portrays the last time step of both

obstacle channel simulation runs. However, this comparison also only serves to validate the simulation code and it can be concluded that the calculations for the obstacle channel scenario are sufficiently accurate.

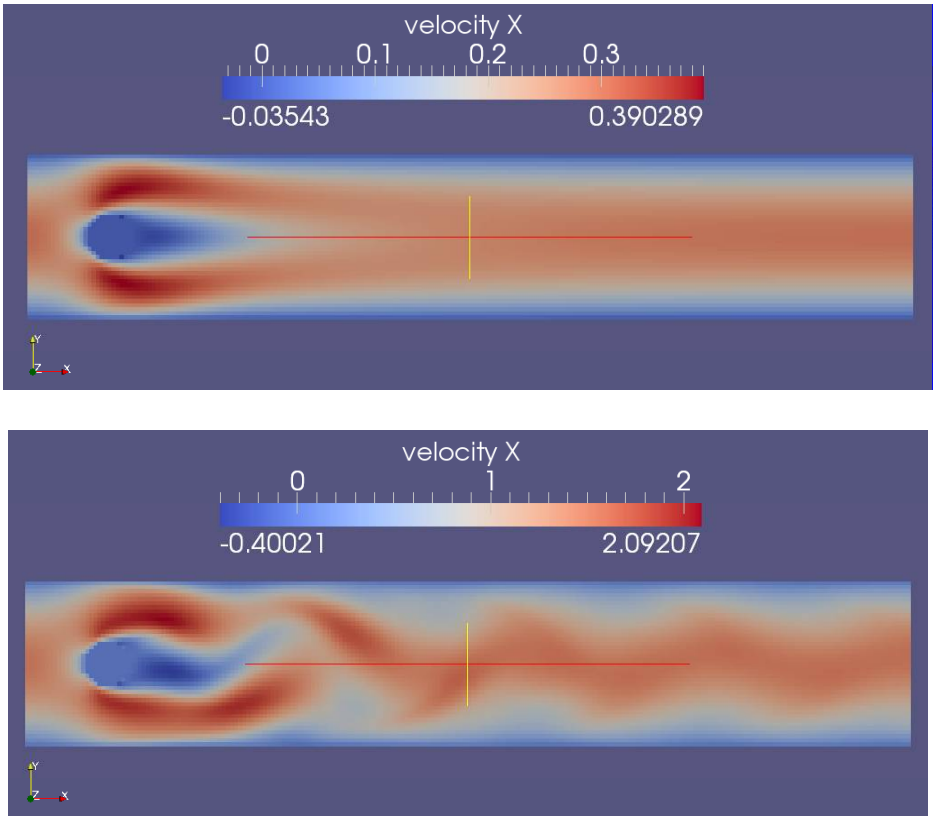


Figure 5.3: Visualizations by Paraview of the velocities in x direction of the last time step of the deterministic run for each of the two obstacle channel setups. First: 220×42 grid with $Re = 20$. Second: 220×41 grid with $Re = 100$.

5.2.2 UQ Results

The unsteady flow scenario is the most interesting case when it comes to the DAKOTA analysis. Table 5.4 represents the results of the UQ studies.

By examining the UQ results, one can see that, even for the unsteady flow scenario, no chaotic behavior can be recognized. Both methods are, at least for the drag coefficient, very close to the deterministic results. The differing outcome of the lift coefficient is the result of the fluctuations in this scenario and the inaccuracy for this small range of values. Also, the SC results do not differ much from the MCS runs, again.

Figure 5.4 and 5.5 illustrate the drag and lift coefficients for $Re = 20$ and $Re = 100$. Again, each plot compares the MCS with the corresponding SC analysis. The similarly

		$Re = 20$	$Re = 100$
		220×41 grid	220×41 grid
MCS	$\mathbb{E}(c_d)$	4.988	3.991
	σ_{c_d}	0.028	0.009
	$\mathbb{E}(c_l)$	10^{-7}	0.59
	σ_{c_l}	10^{-7}	0.089
SC	$\mathbb{E}(c_d)$	4.989	3.991
	σ_{c_d}	0.029	0.01
	$\mathbb{E}(c_l)$	10^{-7}	0.585
	σ_{c_l}	10^{-7}	0.084

Table 5.4: Mean and standard deviation of the drag and lift coefficients c_d and c_l for each of the obstacle channel UQ studies.

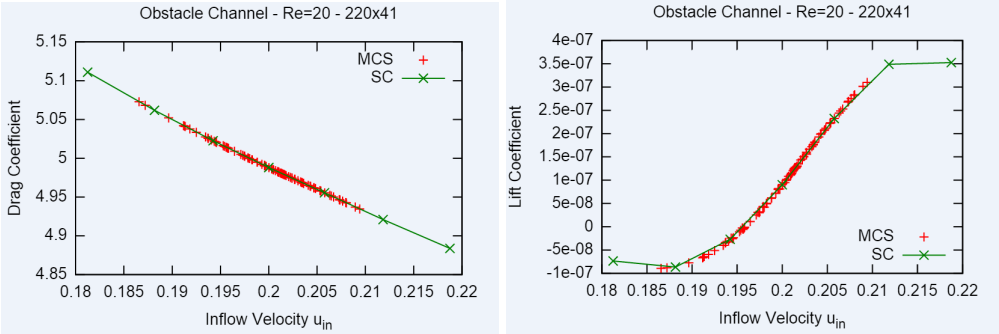


Figure 5.4: Every Sample of the MCS and SC analysis with $Re = 20$: drag coefficient (left) and lift coefficient (left).

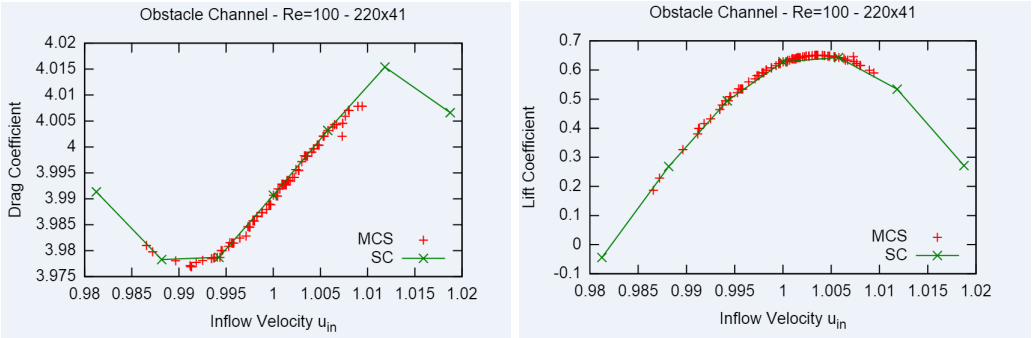


Figure 5.5: Every Sample of the MCS and SC analysis with $Re = 100$: drag coefficient (left) and lift coefficient (left).

results of the different UQ methods, that is analytically represented in the tables above, is once again visualized in the plots below.

The data listed for the two flow scenarios above acts as a proof of concept and further

provides results that can be used as benchmark data. Note that the values are not meant to be used for physical evaluation, since the CFD lab code does not provide the required accuracy. Furthermore, we chose comparatively coarse resolutions because the main objective was to show the functionality of our tool chain and not an exact physical analysis.

6 Conclusion and Outlook

In this thesis, we used several different setups of the backward facing step and the channel with circular obstacle scenario to create a benchmark suite for uncertainty quantification tests using the Monte Carlo sampling and the stochastic collocation methods of the DAKOTA framework. To achieve the desired scenario configurations, we initially had to prepare the CFD lab code. This code modifications enabled us to add an obstacle to an empty flow channel. Additionally we had to implement the computation and output of the relevant quantities of interest. With this background we started the coupling of our simulation code with the analyzing framework DAKOTA. We listed a detailed installation instruction, as well as an explicit description on how to interact with DAKOTA. The usage of DAKOTA did not require any changes to the simulation code itself. The general interface based on reading and writing files is very simple to understand and use. Only the specifications of the DAKOTA input files required more in-depth knowledge, which could be obtained from the very detailed documentation of the framework.

The results presented in Section 5 lead to the conclusion, that the backward facing step scenario is a suboptimal choice for our studies. At least with this setup, our QoI for this scenario, namely the reattachment length, is too robust and does not alternate as desired for an uncertain inflow velocity u_{in}. The reasons for this robust behavior are, on the one hand, the simplified implementation of the reattachment length, which is limited to multiples of the cell width δx and, on the other hand, the coarse grid resolutions that we used for the studies. Therefore, the UQ analysis for the backward facing step is not particularly informative. However, the obstacle channel is a considerable scenario for UQ tests, since the drag and lift coefficients are fluctuating significantly for disturbed input parameters. Further, our aim was not to analyze the physical behavior of the different scenarios. Rather, we planned and succeeded in coupling the CFD code with DAKOTA and collecting a bunch of benchmark data form the UQ studies, that can be used for future work. In addition, the results demonstrate quite clearly that, especially for computationally intensive simulations like the CFD lab code, stochastic collocation methods are considerably superior to sampling methods. Even though we focused on simple and coarse scenarios, MSC took several days longer than the corresponding SC method, in the worst case.

Future work should concentrate on the obstacle channel or further flow scenarios for which meaningful output can be expected. Furthermore, the usage of a finer grid resolution is also conceivable. For improvement of the MCS analysis, it is recommended to specify substantially more samples. We had to define an upper limit of 100 samples, due to the high computational effort of the simulation code. Additionally, a higher

dimension of the input space would lead to more meaningful results, but due to the computational intensity of the numerical solver, an expansion in the dimension would have been gone beyond the scope of this thesis.

List of Figures

2.1 Laminar vs turbulent flow 5
2.2 Staggered Grid 8
2.3 Grid with Ghost Cells 9
2.4 Sample configuration file 11
2.5 Sample Paraview Visualization 12

3.1 DAKOTA "black-box" Interface 20
3.2 DAKOTA Specification Block Relationship 20
3.3 Sample DAKOTA Input File (Rosenbrock) 22
3.4 DAKOTA Parameter and Results File 22

4.1 Backward facing Step Geometry Configuration 26
4.2 Backward facing Step Configuration File 27
4.3 Obstacle Channel Geometry Configuration 28
4.4 Obstacle Channel Configuration File 29
4.5 Output Folder Content 30
4.6 Backward Facing Step Wrapper Script 33
4.7 Obstacle Channel Configuration File Template 36
4.8 DAKOTA Input File - MCS backward facing Step 37
4.9 DAKOTA Input File - SC Obstacle Channel 40

5.1 Visualization backward facing Step 44
5.2 Backward Facing Step Plots 46
5.3 Visualization Obstacle Channel 47
5.4 Obstacle Channel Re=20 Plots 48
5.5 Obstacle Channel Re=100 Plots 48

List of Tables

5.1 Backward Facing Step deterministic Results 43
5.2 Backward Facing Step UQ Results 45
5.3 Obstacle Channel Detertministic Results 46
5.4 Obstacle Channel UQ Results 48

Bibliography

[Ada+14a] B. M. Adams, M. S. Ebeida, M. S. Eldred, J. D. Jakeman, L. P. Swiler, J. Stephens, D. M. Vigil, and T. M. Wildey. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 Developers Manual*. Tech. rep. 2014.

[Ada+14b] B. M. Adams, M. S. Ebeida, M. S. Eldred, J. D. Jakeman, L. P. Swiler, J. Stephens, D. M. Vigil, and T. M. Wildey. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 Reference Manual*. Tech. rep. 2014.

[Ada+14c] B. M. Adams, M. S. Ebeida, M. S. Eldred, J. D. Jakeman, L. P. Swiler, J. Stephens, D. M. Vigil, and T. M. Wildey. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 Theory Manual*. Tech. rep. 2014.

[Ada+14d] B. M. Adams, M. S. Ebeida, M. S. Eldred, J. D. Jakeman, L. P. Swiler, J. Stephens, D. M. Vigil, and T. M. Wildey. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual*. Tech. rep. 2014.

[Bal+14a] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, K. Rupp, B. Smith, and H. Zhang. *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.5. Argonne National Laboratory, 2014.

[Bal+14b] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, K. Rupp, B. Smith, and H. Zhang. *PETSc Web page*. `http://www.mcs.anl.gov/petsc`. 2014.

[Bal+97] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries." In: *Modern Software Tools in Scientific Computing*. Ed. by E. Arge, A. M. Bruaset, and H. P. Langtangen. Birkhäuser Press, 1997, pp. 163–202.

[GDN98] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerical Simulation in Fluid Dynamics*. SIAM, 1998.

[KCD11] P. Kundu, I. Cohen, and D. Dowling. *Fluid Mechanics*. Academic Press, 2011.

[Men14] F. Menhorn. "Uncertainty Quantification in Incompressible Flow using Sparse Grids." MA thesis. 2014.

[Nec05] T. Neckel. "Einfache 2D-Fluid-Struktur-Wechselwirkungen mit einer cache-optimalen Finite-Element-Methode." MA thesis. 2005.

[Nec09] T. Neckel. "The PDE Framework Peano: An Environment for Efficient Flow Simulations." available in Verlag Dr. Hut, ISBN 978-3-86853-147-3. PhD thesis. Institut für Informatik, Technische Universität München, June 2009. ISBN: 9783868531473.

[Ros60] H. H. Rosenbrock. "An Automatic Method for Finding the Greatest or Least Value of a Function." In: *The Computer Journal* 3.3 (1960), pp. 175–184.

[Sch+96] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher. "Benchmark Computations of Laminar Flow Around a Cylinder." English. In: *Flow Simulation with High-Performance Computers II*. Ed. by E. Hirschel. Vol. 48. Notes on Numerical Fluid Mechanics (NNFM). Vieweg+Teubner Verlag, 1996, pp. 547–566. ISBN: 978-3-322-89851-7.

[Smi13] R. C. Smith. *Uncertainty Quantification: Theory, Implementation and Applications*. SIAM, 2013.

[Xiu09] D. Xiu. "Fast numerical methods for stochastic computations: a review." In: *Communications in computational physics* 5.2 (2009), pp. 242–272.

[Xiu10] D. Xiu. *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press, 2010.

[XK04] D. Xiu and G. E. Karniadakis. *Supersensitivity due to uncertain boundary conditions*. 2004.